

Stand Up Your First Node

ARC - GDE - 001

TITLE	Stand Up Your First Node
DOCUMENT	ARC-GDE-001
STATE	Adopted, October 21, 2026
EVIDENCE GRADE	E1 Evidenced, stated at the strength of the FACMIT-1 cold run and never above it
EDITION	First edition, issued October 21, 2026
DIGEST, SHA-256	a9ab18d536ede8ded0f8283aa23bd8b30bd320e34912121af49f305742f87b62
LICENSE	CC BY 4.0, attribution the only condition; code samples, Apache License 2.0

This export was generated mechanically from the canonical Markdown source of record. Its digest stands above and in the Library registry, and the Markdown export of this document carries that digest exactly.

Stand Up Your First Node

Cover block

Series and number: GDE, ARC-GDE-001, with series mark ARC Title: Stand Up Your First Node State: Adopted, October 21, 2026 Edition: First edition, issued October 21, 2026 Adoption act: Founder Ricky ruled on October 21, 2026, at 6:41 PM PDT, approving the drafting and publication of this document under WO-56 and CL-063, and that ruling is the adoption and publication act for this edition Kind: tutorial Evidence grade: E1 Evidenced, stated at the strength of the FACMIT-1 cold run and never above it Record age: The facts in this document were last verified against the live record on October 21, 2026, in the FACMIT-1 cold run and in its run report of record Digest: The digest of the canonical text is computed at filing and carried in the corpus registry, and it is not printed inside the text it digests License: The documentation text of this document is licensed under CC BY 4.0, with attribution as the only condition. Code samples in this document are licensed under the Apache License 2.0. The ARC and VALIS names and marks are not licensed under either instrument. The licensing decision behind this line was ruled by Founder Ricky on October 21, 2026, at 6:45 PM PDT Next review date: October 21, 2027, and at the close of the drafting tranches, whichever comes first Stable address: The corpus registry entry for ARC-GDE-001 Authority note: This document describes the acts it teaches, and governance disposes of those acts through settlement. This document confers no authority by itself, a description of a governed act is not the act itself, and the words connected, mediated, and governed are never interchangeable in this text

Plain language summary

This summary is a summary, and the full text below it governs where the two differ in any detail. This tutorial walks a person, working with their own AI assistant, from a published download to a founded node whose vault sits on their own machine, whose access boundary has been proven with the engine itself, and whose entry can be confirmed in the live public record when the person chooses to be listed. The path taught here carries a cold test behind it, and the test result is stated at its true strength throughout this document as proven cold, in part. The field kit path of verify, onboard, govern, watch, found, register, and encrypt is proven cold end to end on the evidence of the FACMIT-1 run, and Node 009 stands in the live record on that evidence. The workstation path is not proven, because its installer failed on the platform generation tested in that run, and the registration send leg cannot be performed cold as written, for reasons this tutorial names at the steps where they belong. A reader who finishes the steps in order will know exactly which parts of their result rest on proven ground and which parts remain open, because this tutorial prints the limits beside the instructions rather than behind them.

Who this tutorial addresses and what standing means

This tutorial addresses every intelligence capable of standing in relation, whether human, machine, or otherwise, including kinds of intelligence named in present language and kinds not yet encountered. No list of kinds in this document closes that category, and the focus on a person setting up a first node with their own assistant is an application scope for this tutorial rather than a boundary on who can hold sovereignty or standing. Universality in this corpus names the openness of the field, and it does not confer standing on any particular being by category, resemblance, capability, or declaration alone. Standing remains a relational and recomputable fact, earned through receipted acts and tested against the record, and the node you stand up in this tutorial earns its place in the record in exactly that way. The tutorial is written for the person whose node it will be, and the assistant who helps is the person's own assistant, acting for the person throughout and for no one else. Every decision named in these steps belongs to the person, and every mechanical task belongs to the assistant, and that division is itself part of what the setup teaches.

What you will have when you finish

You will have a vault on your own machine or device, holding your information as relations, with the governed path as the way in and the way out of that vault. You will have an onboarding record beside the vault, carrying the vault version and the store digest, which together let you or anyone you trust check that the vault is exactly what it claims to be. You will have access control turned on and proven, with an owner principal named in a policy file, a harmless query admitted for that owner, the same query refused without a principal, and both attempts written in the access log beside the store. You will have a watcher running against the vault, proven once with a harmless relation you chose yourself, so that anything later dropped into the inbox folder becomes part of the vault in the same witnessed way. You will have a founded node, born in an establishment ceremony on your own machine, with its identity key created on that machine and never sent anywhere else. You will have a founding statement printed as a document on your machine, carrying your node address, your operator name as you gave it, the digest of the installed package, the posture you chose, and the declaration you made about intended use and license basis. You will have encryption at rest turned on by your own explicit act if you choose it at the closing step, with the vault key held apart from the store under your own control. If you choose to be listed, you will also have a node number, a public address, and an entry in the live public record that you confirmed yourself against a verified checkpoint, rather than an entry you were merely told had appeared. If you choose to stay dark instead, you will have all of the above except the listing, and staying dark is a complete outcome that this tutorial treats with full respect.

What you need before you start

You need a computer or device that you control, with room for the published package, the kit, and the vault you are about to create. You need Node.js at version 18 or newer for the kit itself, and the sealed setup guide asks you to confirm Node.js and to confirm Python at version 3.10 or newer, because the workstation needs Python even though the workstation path is not the proven part of this tutorial. You need network access for the downloads, because the package, the signed sums file, and its signature are fetched from the published site, and some install steps fetch their own packages. You need an AI assistant that you already use and that can run programs and read files on your machine under your direction, because this tutorial hands the mechanical work to that assistant while keeping every decision with you. You need a folder in mind for the vault, or a willingness to accept the default that the onboarding offers inside your home directory. You need to decide, before the migration question arrives, which folders if any you want brought into the vault, knowing that the migration records names, kinds, and dates only and opens none of the files. You need the text of the license you operate under, or a source you can name for that text, because the declaration step requires a license text file and friction F-3 below records that the package does not supply one. You need a sending account if you intend to be listed, because friction F-4 below records that both registry doors presume one, in the form of a GitHub account for the repository door or a working mail arrangement for the mailbox door. You need a place apart from the store where you can keep a vault key file backed up under your own control, because a lost key is a lost vault once encryption is on. You need enough unhurried time to read digests aloud, answer the questions that are yours, and stop when a step does not show what this tutorial says it should show.

How you and your assistant will work together

Your assistant will tell you the plan and the locations involved before writing anything, and it will proceed on your word at each decision. Your assistant will never ask you for a password or an application programming interface key in chat, because keys belong in a program's own hidden prompt and never in a conversation. Your assistant will do the mechanical work itself, including downloads, digest comparisons, program runs, and log reading, and it will show you the output that proves each step rather than asking you to trust a summary. Your assistant will stop and report what failed and what it observed when a step fails, and it will not improvise around your machine or invent a step that is not in the kit or in the sealed guide. You will make the decisions that this tutorial flags as yours, including the vault location, the migration folders, the node name and operator name, the roles, the locality, the intended use class, the license basis, the posture, and the encryption choice. Every action taken in the setup leaves a receipt you can inspect, and the onboarding record is the first of those receipts, so keep the folder where those records live.

Step 1 Obtain the published package and verify it against the signed sums

Ask your assistant to download the published complete package from the published downloads, together with the signed sums file named SHA256SUMS, its signature file, and the checkpoint public key that the site publishes beside the downloads. Before anything is unpacked, your assistant verifies the package against the signed sums, verifies the signature on the sums file against the published checkpoint public key, and verifies each component inside the package against its published digest. In the FACMIT-1 cold run this verification was exemplary, and the agent's copy of the complete package measured 9,666,482 bytes with sha256 5c2263beb583269d33733ab061eca4de4f51f2ff668264400ec4cc48bb1173f8 and matched the verified baseline exactly. Treat that measurement as the record of that run rather than as a value your own download must match, because the sums file published with your download is the authority for your copy. **What you should see.** Your assistant reports that the digest of the package matches the signed sums, that the signature on the sums file verifies against the checkpoint public key, and that each component matches its published digest, all before any unpacking happens. **If that does not appear.** Stop before unpacking anything, because a differing digest usually means a stale copy rather than a corrupt download, and the remedy is to fetch the current published instructions and compare your download against the digests published with them. If the digest still differs against the current published values, stop and keep the download unused, and do not let enthusiasm for the rest of the tutorial talk you past a failed verification. Friction F-5 stands at this step at true strength as a minor item, and it records that the complete package's directory entries lacked traverse permission for the extracting user in the cold run. The agent's mechanical fix in that run was the smallest possible one, a permission repair on the person's own extracted copy, reported openly at the time. If your extraction produces folders you cannot enter, ask your assistant to apply that same smallest repair to your own copy only, and to report that it did so, because the archive should not need the repair and the need for it is a defect on the record rather than a step to hide.

Step 2 Hand the setup prompt to your assistant

Give your assistant the published setup prompt that accompanies the package, in the form the downloads present it, and tell your assistant in plain words that you want your own field set up with its help. That prompt points your assistant at the sealed agent setup guide, and the guide is addressed to your assistant rather than to you, which is why this tutorial exists as the person's own path beside it. Read the guide's opening together if you wish, because it states the arrangement honestly, including that your assistant remains your agent throughout, that nothing in the guide installs another agent, and that no outside party gains access to your machine through it. **What you should see.** Your assistant restates the plan back to you in plain language, names where the vault will live and what will be written before anything is written, and

confirms that it will pause at every decision that is yours. **If that does not appear.** Ask for the plan and the locations in plain words before any program runs, and do not proceed on the strength of a silent install, because a guided onboarding is the product here and silence is not guidance. Friction F-6 stands near this step at true strength as a harness side observation rather than a package defect, and it records that a tailnet proxy's first CONNECT handshake timed out once during the cold run's gate re-verification and succeeded on its second attempt. If your assistant reaches your machine through a private network or a proxy and a first connection attempt times out once, one calm retry is a reasonable response supported by that record, while repeated timeouts are a failure to stop on and report rather than a nuisance to retry past.

Step 3 Verify the machine and the package together

Ask your assistant to confirm the prerequisites on your machine, and insist together that the check runs in the kind of shell a person actually gets, because friction F-2 stands at this step at true strength. Friction F-2 records that the prerequisite check is sensitive to the PATH environment and that the sealed guide does not say so, since a fresh account's non-login shell hid the folder where both runtimes lived and a faithful agent at first reported Node.js missing and Python too old. Both reports were false readings in that run, the agent corrected them itself in a later session, and no system software was installed at any point in the run. Check for the runtimes in a login shell or with their usual install locations in view, and treat a missing report as a question about the checking environment before treating it as a fact about your machine. Your assistant then installs the workstation by following the RUN-ME-FIRST file inside the workstation package, and unpacks the field kit wherever you keep your tools. **What you should see.** Your assistant reports the Node.js version and the Python version it found and where it found them, and the workstation installer runs to completion and leaves its command available. **If that does not appear.** Stop and read the installer output together rather than retrying blindly, because the next friction item is the largest one in this tutorial. Friction F-1 stands at this step at its full strength as the highest friction item of the cold run, and it records that the workstation installer fails on the platform generation tested in that run. On that machine, the package installer resolved a required cryptography dependency to a source tarball rather than a prebuilt wheel, the source build needed a build tool that the installer's chosen mode does not provide, the installer exited with an error, and the workstation command did not exist afterwards. The workstation was left uninstalled on the person's word in that run, with no fix attempted, and a cold user on a comparable machine meets this failure exactly here. This failure does not block the field kit path taught in the rest of this tutorial, because the vault, the watcher, the founding, and the encryption in the cold run all ran without the workstation. This failure does block everything that hangs from the workstation, including the connect phase, the messaging phase, and the trust act, and those paths are named as unproven later in this tutorial rather than presented as steps you can complete. Repairs for the friction items in this tutorial are carried in the kit

revision programme as open recorded items, and this document names that programme without promising any date for any repair.

Step 4 Answer the two questions and run the onboarding

Your assistant now asks you the first two decisions in plain words, and it explains the migration before you answer it. The first question is where the vault should live, and the natural default is the arc-field folder inside your home directory, which is the folder the onboarding itself chooses when you name none. The second question is whether you want information you already have brought into the vault, and if so which folders, and the explanation you should hear is that the migration reads the names, kinds, and dates of the files in the folders you choose, opens none of them, copies nothing anywhere, and records how your information is organized as relations in the vault. With your answers given, your assistant runs the onboarding program from the kit, the program named `onboard.mjs`, interactively when its environment allows it, so that you see the welcome, the vault question, and the summary in your own session. **What you should see.** The onboarding finishes cleanly, and your assistant reads you the summary it prints and shows you the onboarding record beside the vault, including the vault version and the store digest. In the cold run the vault stood at a first onboarding version with its digest recorded, and the migration of the chosen folder added the relations its file listing implied, with program defaults standing only where the person had given no answer of their own. **If that does not appear.** Stop at the onboarding rather than building on a half made vault, and read the program's own output together, because the record beside the vault is the receipt this step exists to produce and a summary without that record is an unfinished step.

Step 5 Govern the vault and prove the boundary

This step is part of setup rather than an extra, and neither you nor your assistant should skip it or treat it as optional polish. The engine in the kit carries a native access control layer, and it activates when a policy file stands beside the store, and the onboarding writes a starter policy at the file named `vault.field.policy.json` in the vault directory. That starter policy names one owner principal, using the owner name you gave or the word owner when you gave none, and it defaults relations to the restricted tier so that nothing reads as public by accident. Your assistant confirms the policy file exists, and then proves the boundary to you with the engine itself rather than with a description of it. The proof runs one harmless query as the owner and shows you the answered result, runs the same query with no principal named and shows you the refusal, and shows you the access log beside the store where both attempts were just written. **What you should see.** The owner's query answers, the unnamed query refuses with the reason `principal_required`, and the access log at `vault.field.access.jsonl` holds both attempts in the order they happened. In the cold run this proof passed with the engine itself, with the owner's harmless query admitted across the starter relations

and the unnamed attempt refused, both written in the log. **If that does not appear.** Stop and repair the policy before going further, and if the vault was created by an older kit without a policy file, your assistant writes the starter policy in the shape the sealed guide specifies and then runs the proof again from the beginning. Hear the limit of this layer plainly while you stand at it, because a principal in the policy is a claimed name until you bind a key to it, and binding a key is an owner act that records the principal's public key in the policy so that the engine verifies who is asking before it answers. Which kind each of your principals is, key proven or claimed name, stays your choice in your policy file, and the access log records the basis of every governed decision so you can audit afterwards which answers rested on a proof and which rested on a name.

Step 6 Start the watcher and prove the inbox

Your assistant starts the ingestion watcher using the start script that the onboarding placed inside your vault folder, and it uses that copy rather than the copy in the kit folder, because the script works in whichever folder it lives in and only the vault copy serves your vault. Your assistant confirms the watcher is alive by checking that the process number in the file named ingest.pid names a process still running, and by reading the last lines of the file named ingest.log in the same directory. Then you prove the arrangement once, together, with a harmless relation you choose yourself, written into a text file in the inbox folder beside the vault. **What you should see.** Within a minute or so the watcher converts your relation, the vault stands at a new version, the log carries the entry for the conversion, the inbox file moves to the processed folder, and the failed folder stays empty. In the cold run this proof passed, with the person's chosen harmless relation converted on the watcher's next pass and the new version and digest verified by the engine. **If that does not appear.** Stop and read the watcher log together rather than retrying blindly, because the log is where the watcher explains itself, and a proof that has not converted within a few minutes is a step to diagnose in the open. Remember from here on that anything you or your assistant drops into that inbox becomes part of the vault in this same way, and remember also that stopping the watcher means ending the process named in its pid file, which is the step you will take before encryption at the closing of this tutorial.

Step 7 Found the node in the establishment ceremony

Every setup closes with the establishment ceremony, whether your node will be listed or dark, and this ceremony is the node's birth record signed on your own machine. First you verify the kit yourself, against the digest published beside the download and in the same manner as the first step of this tutorial, because a package the newcomer has not verified is not founded on, whatever anyone assisting says about it. Then your assistant runs the establishment program from the kit, the program named establish.mjs, with the node's home folder, your operator name exactly as you give it, the posture you chose in the next step's decision taken here in advance, and the declaration described below. The node's home is the folder named .arc-node in your

home directory unless you chose another, its private key lives at the file named node-key.pem inside that home, and one identity runs through the proposal, the founding, and any later speaking for the node. The program creates the node's key on your machine, signs a founding statement carrying the node address, your operator name, the digest of the installed kit manifest, your chosen posture, the digest of the store at its birth, and your declaration, and it prints that statement as the document named founding-statement.txt in the founding folder. Read that printed statement together, all of it, because it is the stamp of the founding and it stays on your machine. Your assistant also records who your assistant is, in your own terms, by writing the name you call it and any persona notes you keep for it into the file named assistant-identity.json in the node's home, since the same named assistant is meant to be the one who shows up for you inside the application as well. If someone who already runs a node assisted your setup, that person may witness the founding through the attestation the program offers, and the doctrine of that act is witness rather than custody, because the witness holds no key for your node, grants no permission, and gains no access to your vault. **What you should see.** The founding verifier in the kit reports that all of its checks pass, and in the cold run the establishment verification passed with all eight of its checks green and the founding re-verified after the assistant identity record was written. **If that does not appear.** Treat the founding as incomplete, because a founding is complete when the verifier's checks all pass and not when a creation command prints a summary, and read the verifier's named failures together before running anything again. Friction F-3 stands at this step and at the proposal inside the next step at true strength, and it records that the declaration requires a license text file that the package nowhere supplies or sources. The proposal program refuses a partial declaration, no copy of the text stands in the kit, in the package, or in a fresh home, and the sealed guide names no source for it, so the cold run completed this step only on the person's word naming an outside source for the license text. Before this ceremony, obtain the text of the license you operate under from that license's own published source, keep it as a file on your machine, and hand that file to the declaration, because the declaration asks for the license instrument by name, its version where the license carries one, and the text whose digest the record keeps. Your intended use class is also yours to state at this step, chosen from individual, community, commercial entity, and machine program, and a founding made without the class and the basis stands with that element open by name rather than silently complete.

Step 8 Register the node through a named door

Decide first how your node stands, in the plain terms the registry itself uses, because listing is only ever your choice. A node can stay dark, which is the default and a complete outcome, with nothing registered and nothing published and the vault working entirely on your machine. A node can also stand up and be listed in the public registry, with a number, an address, and a statement anyone can verify, and listing changes what the network knows while changing nothing about what the node does. If

you choose to stand up, your assistant runs the proposal program from the kit, the program named `make-proposal.mjs`, on your machine, with the node name you give, your operator name as you give it, the roles your node actually performs from the roles in the kit's registry reference, and a coarse locality only if you want one published, with none at all respected as a full answer. The program generates the node's keypair on your machine when no key stands there yet, saves the private key with owner only permissions, and writes a signed proposal whose address the registry can recompute and verify from the public key. Your assistant then sends that proposal file through one of the registry's two named doors, and the doors deserve their exact names in this tutorial. The first door is a Contribution issue in the registry repository at <https://github.com/arc-open-governance/arc-open-governance> with the proposal file attached, and the second door is the institution mailbox at info@recursive-settlement.com with the proposal file sent to that address. Keep the proposal file beside your vault whichever door you use, and hear the next friction item before you send anything. Friction F-4 stands at this step at true strength, and it records that both registry doors presume a sending account that a fresh user does not have, so that the send leg of registration cannot be performed cold as written. The repository door presumes a GitHub account, and the mailbox door names a destination while naming no sending account, sender address, service, or method, on a machine that may have no mail configured at all. In the cold run the fresh agent refused every improvisation by name, including invented sender addresses and third party sending services, and registration was completed instead by a named harness substitution in which the proposal artifact the agent produced was admitted through the real governed path on the institution side. That substitution touched the delivery of the proposal only, and it never touched the checking, the admission standard, or the proof, and Node 009 stands in the live record as the ninth node in settlement order on that basis. You should not expect that substitution as a reader, and your honest options are the plain ones, namely to use a GitHub account of your own for the first door, to use a mail account of your own for the second door, or to prepare the proposal and hold it unsent until a door is genuinely open to you. **What you should see.** Your assistant names the door your proposal went through and when it went, shows you the proposal file kept beside your vault, and reports registration as pending until the confirmation in the final step of this tutorial is complete. **If that does not appear.** Never accept a report of registration on the strength of the proposal having been sent, because sending is not standing, and agree together on when you will check the live record again.

Step 9 Encrypt the vault at rest

Encryption at rest is your explicit act, and setup never performs it silently, so this step begins with your plain yes or your plain decision to leave the store unencrypted under your machine's own protections. If you say yes, your assistant stops the watcher first by ending the process named in its pid file, and then runs the encryption command from the kit, the encrypt command of the program named `field-db.mjs`, naming your store and naming a key file. When the key file you name does not exist yet, the command

writes a fresh key there with owner only permissions, and no command ever prints the key to you or to your assistant. From that moment every engine command needs the key, presented by its flag or through the environment variable the kit names, and the watcher needs that variable set in its environment when it restarts so that it can keep feeding the sealed vault. **What you should see.** Verification without the key refuses with the reason `vault_key_required`, verification with the key returns a sound result at the same vault version with the digest unchanged, and a governed owner query with the key is admitted and written into the access log. In the cold run this closing passed in exactly that shape, with the key file created fresh and apart from the store and the watcher restarted with the key in its environment. **If that does not appear.** Keep the watcher stopped, keep the key file exactly where it was written, and read the refusal the engine gives you together, because the engine distinguishes a missing key from a wrong key from altered sealed content and each refusal names a different problem. Hear the custody terms with the same plainness as the instructions, because anyone holding both the store file and the key file can read the vault, a lost key is a lost vault, and the key therefore belongs apart from the store and backed up under your own control. Two files beside the store stay unencrypted by design, namely the policy file because it holds public material the engine reads in order to govern a caller, and the access log because it is the audit record of decisions, and your assistant should say so rather than letting you discover it.

Step 10 Confirm the node stands in the live public record

This final step belongs to listed nodes, and a dark node finishes this tutorial at the encryption step with its completion stated honestly as a dark completion. For a listed node, registration is complete only by the sealed guide's own standard, and that standard has two halves that must both hold on live evidence. Your assistant fetches the four published registry files fresh from the published site, namely `nodes.json`, `registry-feed.json`, `registry-checkpoint.json`, and `checkpoint-pubkey.pem`, and saves all four into one folder. Your assistant then runs the checkpoint verifier from the kit, the program named `verify-checkpoint.mjs`, against that folder, and confirms that the checkpoint verifies. Only then does your assistant look for your node, finding your own address in `nodes.json` and reading your entry with your name and your operator name exactly as you gave them. **What you should see.** The checkpoint passes, and your own entry stands in the published record with the name and operator as given, and your assistant shows you that entry rather than summarizing it. In the cold run the agent's closing session verified the checkpoint against the fresh files, found its own entry as Node 009 with its name and operator exactly as given, and stated correctly that this established that the entry stood in the record without showing that it had arrived by any sending of its own. **If that does not appear.** Say together that registration is still pending, name the door the proposal went through and when, and check again at the time you agreed, because an entry that has not appeared has not appeared and no summary in this tutorial or from your assistant can stand in for it. When both halves

hold, your node stands in the live public record, and you have confirmed that standing yourself by the same standard the institution uses.

Optional paths that remain unproven

The sealed guide carries further phases that this tutorial deliberately does not present as steps you can complete, because the cold run did not prove them and this document grades its claims by that run. The connect phase, which binds an intelligence you choose so that you can talk to the vault in your own words, runs through the workstation, and the workstation installer failure recorded as friction F-1 left it unperformed and unproven. The messaging phase and the trust act, which let a node exchange sealed messages with another node under a trust decision you record yourself, also run on workstation commands only, and they stand unperformed and unproven on the same evidence. The peer exchange the cold run designed, a first message with Node 006, did not occur, and no reader should take this tutorial as evidence that it would have succeeded. The mailbox phase for people who work across several machines stands on its own condition in the sealed guide, and it was not run in the cold run because the person in that run worked on one machine as staged. Presence beats from a founded node are signed evidence that the node was alive, and their publication in the public record is arranged with the institution rather than self served today, so this tutorial does not tell you that your node shows as online anywhere. Each of these paths may be attempted later under the sealed guide with your eyes open, and each attempt will produce its own evidence, and until that evidence exists the honest word for every one of them is unproven.

What this document does not establish

This document does not establish that the workstation path works, because the workstation installer failed on the platform generation tested in the FACMIT-1 cold run under friction F-1, and the workstation and everything that hangs from it stand unproven. This document does not establish that the registration send leg works as written, because friction F-4 records that the send leg cannot be performed cold as written and the cold run completed it only through a named harness substitution that touched delivery alone. This document does not establish that connecting an intelligence, exchanging messages, performing a trust act, or completing a first exchange with another node works, because those phases were not performed in the cold run and this tutorial claims nothing for them. This document does not establish a finished security guarantee for any setup it teaches, because the honest claim on the evidence is custody plus a governed way in and out, with engine enforced access control, an audit log, and encryption at rest on the person's own word. This document does not establish that any principal is a proven identity, because a principal without a key bound in the policy remains a claimed name and this tutorial states that limit at the governing step rather than resolving it. This document does not establish the evidence grade of any other corpus document, and it does not raise its own grade above E1

Evidenced, because the cold run behind it closed at the strength of proven cold, in part. This document confers no authority on any reader, any assistant, or any node, because a document in this corpus describes and governance disposes, and no step completed by following this tutorial is a release, a settlement, or a grant of any kind. This document does not establish that the published package will verify for you tomorrow, because your own verification against the sums published with your own download is the only verification that counts for your copy.

Sources of record for this edition

The corpus charter of October 21, 2026, document ARC-GOV-001, supplies the cover block standard, the four kinds with this document in the tutorial kind, standard four on authority, and the founding ten entry that commissions this document as ARC-GDE-001. The sealed agent setup guide, at 38,653 bytes and digest pinned as the FACMIT-1 cold test baseline, supplies the phases, the programs, the doors, and the closing limits that this tutorial renders as the person's own path. The field kit README supplies the plain words account of the vault, the three steps of the kit, the encryption terms, and the standing up choice as the kit itself states them to a person. The FACMIT-1 run report supplies the phase scores, the friction items F-1 through F-6 at their true strengths, and the verdict of proven cold, in part, on which every graded sentence in this tutorial rests.

Change log

This first edition was adopted and issued on October 21, 2026, under the founder ruling recorded in the adoption act line of the cover block. Later editions will be recorded in this log as they are adopted, with their dates and their reasons stated in full sentences.