Security in Nabto Solutions

NABTO/001/TEN/036

1 Contents

2 Abstract 3

3 Important Notes 3

4 Legal Disclaimer 3

5 Bibliography 3

6 Nabto Platform Basics 4

6.1 Nabto Communication Patterns 4

7 Security Overview 5

8 Client Authentication Models 6

8.1 CA signed RSA certificate with user id 6

8.2 Self-signed RSA certificate and local pairing ("PPKA") 7

8.3 Application level authentication on device 9

8.4 Recommendation 10

9 uNabto Device Security 11

9.1 Pre-Installed Id/Key Pairs 11

9.2 Key Installation at Runtime 11

9.3 Factory Reset and Device Recycling 13

9.4 Roadmap feature: Central Custom Authorization 13

9.5 Local Security 14

10 Export/Import Restrictions in US and France 14

11 Appendix A 15

11.1 Secure Device Registration with Basestation 15

11.2 Secure Client Connections to a Device 16

11.3 Encryption and authentication functions in uNabto 17

11.4 Random Functions in uNabto 18

2 Abstract

This document describes security concepts in the Nabto platform and outlines the tasks related to implement a secure solution within the Nabto platform.

3 Important Notes

It is of utmost importance that the content of this document is understood in order to build secure solutions. Moreover**, it can have severe legal consequences if not adhering to the import / export regulations as outlined in section 10**. Also, it is important to understand that the platform provides the features to build a secure solution with little effort - but it still requires care and understanding; ignoring or misunderstanding the provided guidelines in this document can still yield an overall insecure solution, regardless of the platform's general capabilities.

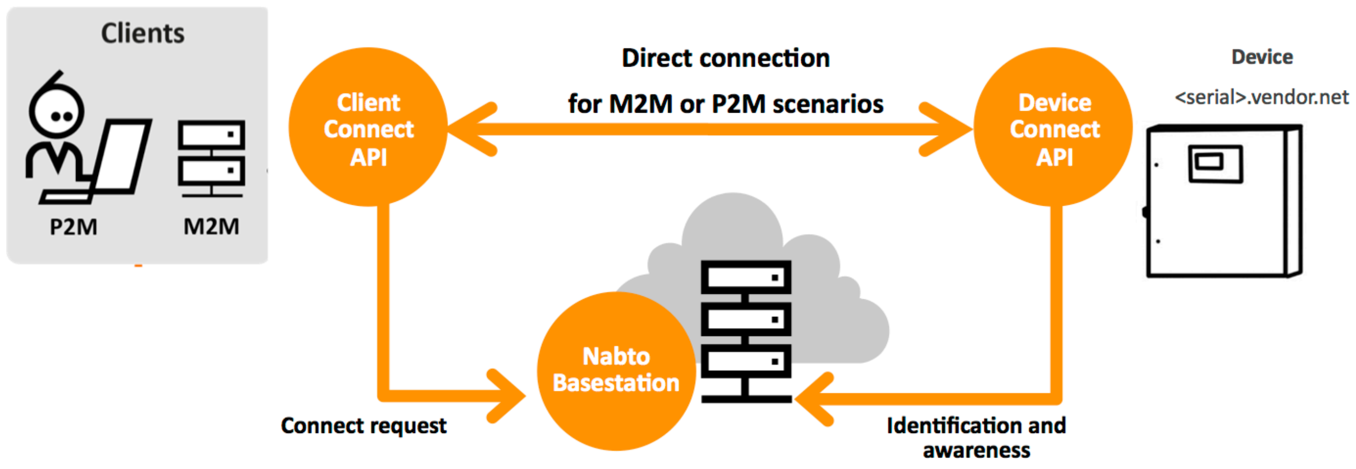
4 Legal Disclaimer

Nothing in this document should be considered legal advice. Please consult a lawyer for actual legal advice, especially relevant with respect to understanding import / export regulations in section 10.

5 Bibliography

TEN023	NABTO/001/TEN/023: uNabto SDK - Writing a uNabto device application
TEN025	NABTO/001/TEN/025: Writing a Nabto API client application

6 Nabto Platform Basics



The Nabto platform consists of 3 components:

- Nabto **client**: Libraries supplied by Nabto, used by the customer's application
- Nabto **device**: The uNabto SDK - an open source framework supplied by Nabto, integrated with the customer's device application
- Nabto **basestation**: Services supplied by Nabto (Nabto- or self-hosted) that mediates connections between Nabto clients and devices.

The Nabto client initiates a direct, encrypted connection to the Nabto enabled device – the Nabto basestation mediates this direct connection: The device's unique name, e.g. <serial>.vendor.net, is mapped to the IP address of the Nabto basestation – this is where devices register when online and where clients look for available devices. After connection establishment, the client and device communicates directly with each other, the basestation is out of the loop – no data is stored on the basestation, it only knows about currently available Nabto enabled devices.

The client can also discover the device if located on the same LAN and communicate directly without the basestation – useful for bootstrap scenarios or for offline use.

Integrating Nabto on the customer's device is the topic of [TEN023].

Nabto client applications are developed using the Nabto Client SDK described in [TEN025]. The Nabto Client SDK is the lowest level way of developing a Nabto application - several wrappers exist on top of this lowest level SDK to provide a more abstract experience, for instance for developing Cordova/Ionic or Xamarin hybrid apps or just simplify native Android and iOS app development.

6.1 Nabto Communication Patterns

The Nabto platform supports 3 communication patterns that will be referenced throughout this document:

RPC: The Nabto P2P-RPC communication mechanism allows a client to securely invoke a remote function on a Nabto device. The device implements an interface definition shared between client and device, the client works with normal JSON documents, exchanged in a compact representation with the device.

Streaming: Nabto P2P-Streaming can be used for retrieving larger amounts of data from a device or sending e.g. a firmware update. With sufficient resources available on the device, Nabto P2P-Streaming can be used for high performance streaming suitable for video scenarios.

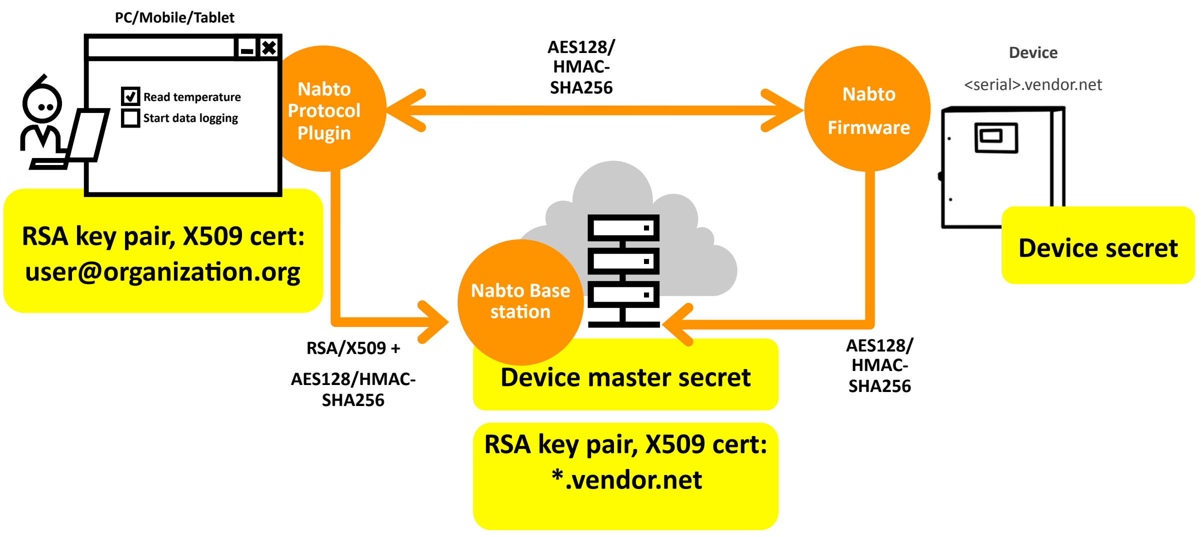
Push: Nabto Push is used for communication initiated by the device, for instance to implement mobile push notifications or to support big data scenarios where data is collected centrally for further analysis. Nabto Push can also trigger an M2M scenario using RPC or Streaming - e.g. when a certain condition is triggered, the device sends a Nabto Push message and a server function invokes an RPC function or streams data.

7 Security Overview

The Nabto platform makes it possible for users to securely communicate with devices – and for device owners to make sure only intended users can access the device.

Security in Nabto is based on the following:

- **RSA key pair to authenticate client towards device**
    - either using a public key signed by CA of choice (e.g., Nabto CA), X.509 subject contains user's verified id
    - or using an unsigned public key and no central authentication but instead using an initial pairing of device and client in a trusted setting
- **RSA key pair to authenticate basestation** (towards client) and establish secure connection to client. Public key signed by CA of choice (e.g., Nabto CA), X.509 subject contains host name pattern associated with basestation, e.g. *.vendor.net. The basestation key and certificate is delivered by Nabto when initially setting up the basestation.
- **Device/basestation shared secret**: Shared with basestation to authenticate device towards basestation (and vice versa) and establish secure connection to basestation and clients. Section 9 describes how to obtain device secrets.
- **Device/client shared secret (optional):** Enable secure offline communication between client and device.



The device/basestation shared device secret approach minimizes the complexity on the device, making the platform able to run on even extremely resource constrained devices such as 8-bit MCUs with a few hundred bytes of memory available. This shared secret approach requires the basestation to be trusted. It is in the Nabto roadmap to add (optional) PKI support to the device – meaning that the basestation only will have a mediating role (assist in exchanging certificates) and can hence be untrusted at the cost of increased complexity and footprint on the device.

Appendix A describes in detail how the keys and certificates are used for securing remote communication within the Nabto platform.

Local (offline) communication between two devices on the same LAN can either be in cleartext or encrypted using a pre-shared secret between device and client. For instance, this can be bootstrapped through a QR code scanned by the client, representing a secret installed at the factory on the device. On the established channel, a client specific secret can then be exchanged. This approach is described in section 9.5.

8 Client Authentication Models

When designing a Nabto solution, you can choose among a few different authentication models, each with its own pros and cons.

8.1 CA signed RSA certificate with user id

This is the traditional Nabto approach to client authentication where the client is identified through an RSA certificate signed by a CA. The certificate has a unique user id in the X509 Subject field, it is up to the CA to populate this after verifying the user's identity when issuing the certificate. For instance, the user's email address can be used as id, the CA verifies ownership of the address when issuing the certificate by sending an email with a link containing a unique token to be clicked. Or the CA can use an existing database of validated users or a directory service.

At connect time, a regular SSL handshake takes place between the client and the basestation (see section 11.2 for details). The authenticated client's identity embedded in the client certificate is securely passed on by the basestation to the device on the secure channel between these two as part of the connect request. The device then authorizes the user to proceed with the connection and invoke the requested functionality - based on an access control list on the device, mapping user identities to privileges.

Pros:

- Possible to use existing identity management infrastructure.
- Devices are paired with *users* not client *devices* (as opposed to the approach described in section 8.2), meaning that a user can access a device from many different clients without re-pairing.
- Possible to centrally define access control lists on devices based on users' identities.

Cons:

- For residential users and small businesses, the user experience becomes cluttered as there is a signup step to verify the identity. For larger organizations / projects with an existing identity management infrastructure, this is less of an issue.
- A central user management service must be provided and maintained. Again, for larger organizations or projects such probably already exists. Nabto no longer provides such services and recommends customers without their own existing user management services to use the PPKA approach described in the next section.

Section 7 "Access Control" in TEN023 describes how to enforce access control on the device. The clientId as mentioned in that section is the id embedded in the signed client X509 certificate when using this client authentication model.

Section 5.5.1 "CA signed certificates" in TEN025 describes how to use signed certificates in the client.

8.2 Self-signed RSA certificate and local pairing ("PPKA")

Instead of relying on the user's identity being embedded in a signed certificate and the basestation validating this certificate at connect time, an alternative option is provided that eliminates the need for central user management, simplifying infrastructure and user experience in some scenarios. We denote this "Paired Public Key Authentication" (PPKA).

Instead of the device maintaining a list of ids of authorized users in its access control list and comparing this with the user's identity received from the basestation (as with the CA signed certificate approach, outlined in section 8.1), the device receives just a fingerprint of the client's RSA public key on the encrypted channel between device and basestation at connect time. The access control list on the device then contains a set of RSA public key fingerprints obtained in a trusted setting, e.g. on a local network and/or after the device has been put temporarily in local pairing mode (similar to WPS bootstrapping of residential network routers).

This greatly simplifies the overall solution for scenarios where the vendor just wants to allow users to access their specific device. No need for a user management portal or maintaining any central knowledge on users - the users just interact with their product and perform secure pairing locally with it.

Nabto provides a full implementation as part of the client starter apps and the uNabto SDK for pairing clients and devices, maintaining the ACL and enforcing access control based on the public key fingerprints.

For sufficiently capable devices[1], the access control list on the device can also be populated using some custom service that manages identities and centrally pairs user identities with public key fingerprints - making it possible to achieve the central management that is one of the major benefits of the CA signed approach, getting the best of both worlds.

Pros:

- Simplified overall solution with no need for central user management infrastructure.
- Simpler user experience with no need to sign up for a service to pair with device.
- Simpler user experience as the user need not enter a username/password but just specify a display name for the RSA keypair (with a sane default of the client device's name ("Alice's iPhone SE")).
- Typically much better security than the username/password based approach described in section 8.3 as there is a risk with that solution of users not picking sufficiently strong passwords.
- Directly supported by the Nabto framework for both the streaming and RPC communication patterns (as opposed to the username/password based approach described in section 8.3 where the application level interface definition must be modified to pass on credentials with each RPC requests)
- Possible to build a custom solution that adds centrally managed user identities for devices sufficiently capable of communicating with such service (e.g., capable of running a full https stack).

Cons:

- The default implementation currently requires the client and target devices to be co-located on the same network to pass on the public key fingerprint from client to target device. It is in the nearterm roadmap to add remote pairing by adding an authentication step at pairing time, e.g. through a pin code provided by the device owner / administrator.
- The default implementation pairs client devices, not users, with the target device - meaning that users will have to pair all their devices (e.g. both a smartphone and laptop as individual clients, each with unique public key fingerprints). A central identity management service that maps user identities to device public keys would remedy this.

Section 7 "Access Control" in TEN023 describes how to enforce access control on the device in general terms. For the specifics regarding the RSA fingerprint based access control, consult the README file in the RSA fingerprint access control module source dir unabto/src/modules/fingerprint_acl.

Section 5.5.2 "Self-signed certificates" in TEN025 describes how to use self-signed certificates in the client.

[1] meaning that the device is capable of securely communicating with the service, e.g. using a standard HTTPS stack - something possible on any Linux based device but that will likely pose a problem on the most limited systems

The Nabto heat control demo solution provides a fully working example of both client and device implementation - see https://github.com/nabto/ionic-starter-nabto and https://github.com/nabto/appmyproduct-device-stub for client and device, respectively. To sign up for a free trial account to test the software, please go to https://console.cloud.nabto.com.

8.3 Application level authentication on device

Applications may implement their own means of authentication, bypassing Nabto's built-in RSA key/pair based mechanisms described in the preceding sections. This is typically used in tunneling/streaming applications where an existing HTTP or RTSP based application is accessed through a Nabto tunnel and the client uses HTTP or RTSP basic authentication on top of the Nabto tunnel.

Hence, Nabto provides security similar to a typical HTTPS connection: The basestation's identity is verified through its RSA/X509 certificate while the client is anonymous at the Nabto level using a self-signed certificate with no significance attributed to the public key fingerprint as described above for the PPKA approach and with no identity embedded in the certificate as for the CA based approach.

Communication between client and basestation/device is still encrypted and is hence confidential with integrity checked on a secure channel established to the device based on its shared secret. But client authentication is performed by the existing application by verifying e.g. a user specified username / password combination.

To prevent brute-force attacking users' passwords, a strict password policy must be enforced requiring passwords of reasonable length, e.g. at least 10 characters. This should be combined with an exponentially increasing period of disabling login on wrong input after a few failed attempts.

Pros:

- Possible to leverage an existing authentication solution already present on the target device, especially relevant if Nabto is just proxying for an existing service, e.g. using HTTP/RTSP basic authentication.
- Possible for the device owner (administrator) to create new users with sharable credentials through alternative channels (e.g. communicate a username and password by voice or on a piece of paper). Users may be created proactively, i.e. no need for a keypair to first be created on the client device.

Cons:

- Potentially very insecure unless using strong passwords and blocking of login attempts after failure. The device is to be considered exposed to the open internet with this approach so it is of utmost importance to educate users about the need for picking strong passwords or using auto-generated random strong passwords.
- Does not work well with the Nabto RPC mechanism; credentials (or a session token) must be supplied by the client with each request and modelled into the RPC interface definition and verified at each request invocation on the device.

To use this approach despite its shortcomings, use a self-signed client certificate as outlined in section 5.5.2 of TEN025. In the uNabto device, always allow access at the Nabto framework level (section 7 in TEN023) - and then make sure in your application to enforce your own means of authentication and authorization.

To use this self-signed certificate, a basestation of Nabto version 3.0.15 or newer must be used. For older basestation versions, a dummy Nabto CA issued "guest" certificate can be used, available for download in the resource bundle from https://www.nabto.com/downloads.html.

8.3.1 One-Time Password Support

Authentication and authorization at the application level can also be implemented through the use of one-time passwords: In some scenarios one-time passwords are a simple and practical solution to provide temporary access to a device. For instance, this can be used to grant temporary remote access to a device in a support situation without updating access control lists.

This can be implemented at the application level when handling Nabto RPC requestss: The vendor's client application passes along a code in the device request that the device validates – if ok, the request is executed even if the user is anonymous and/or does not have the privileges that are usually necessary.

An example of such a scenario could be a device equipped with a display that can show a 4 digit code. When the user presses a "support" button, a code is generated that the user reads aloud to the hotline personnel. The hotline personnel invokes a function on the device and passes along the code the user told on the phone. The device validates the code against the one shown in the display. After some short time (and perhaps after the first execution), the code is expired.

8.4 Recommendation

If you have an existing identity management infrastructure and you need to use these identities to control access on the device, the approach in section 8.1 with identities embedded in CA signed certificates is the optimal approach. This will require integrating the Nabto CA services with your infrastructure - contact Nabto support for further information in this regard.

If you don't need or want to manage user identies centrally and just want users to be able to interact with their devices, the self-signed public key based approach of section 8.2 is recommended due to its simplicity and high level of security.

If you have an existing credentials database on the target device, the application level approach of 8.3 can be considered. However, due to the great risk of using weak credentials, we highly recommend investigating the alternative options described above, perhaps combining the PPKA approach with the existing credentials database.

The onetime password approach described in section 8.3.1 can be combined with any of the other approaches described.

9 uNabto Device Security

Devices based on the uNabto SDK currently supports security through a shared secret approach – it will later be possible to choose a PKI approach as is already used for clients. The shared secret is either derived from device id (e.g. <serial>.p2p.vendor.net) + a master secret known by the basestation or it is explicitly defined for the individual device.

For evaluation and development, the device management portal on https://console.cloud.nabto.com allows creation of a number of free id and crypto key pairs. Additional pairs can be purchased using a credit card in the portal. If using a self-hosted basestation for production, keys will be issued by Nabto either through a webservice available to the customer or through pre-generated lists.

How the device id and shared secret is made available to the uNabto framework when starting it up is completely up to the vendor to decide, it must just somehow be present when starting up.

The conceptually simplest approach is where the device id and shared secret are pre-installed on the factory and simply passed to the uNabto framework when starting up (described in section 9.1 below). However, while simple, this approach has certain drawbacks: It requires the installation of unique information onto each device instead of just shipping identical firmwares. And handling the very sensitive device cryptographic keys might not be suitable for the manufacturing process. Also, the vendor might want to "activate" the device by allowing the user to purchase the device key. Suggestions for more flexible installation methods are provided in section 9.2.

9.1 Pre-Installed Id/Key Pairs

With this simple approach, the shared key is installed on the factory. Hence, it does not require the more complex client application or advanced embedded platform capabilities (HTTPS support) as well as central service interaction as needed for the more flexible approach described in section 9.2. This means it works well with very constrained platforms - and is also simpler to work with when prototyping and developing.

The key is passed on to the uNabto framework during initialization, specifically it must be set in the `presharedKey` field of the `nabto_main_setup` struct, accessible through `unabto_init_context()` (see TEN023 "uNabto initialization" for details).

How the key is being passed to the struct is of course up to the device integrator to decide. The default uNabto tunnel demo application supports getting the key passed on the commandline – in many tunnel use cases this default application is used and a wrapper script reads the device key from the device's file system.

9.2 Key Installation at Runtime

Instead of installing a permanent device key in the factory, the key can be installed on the device at runtime from a central key generation service at the cost of overall more complex system functionality. We denote this runtime device provisioning.

9.2.1 Device based provisioning at the end-user

If the device is sufficiently capable to run an HTTPS client, you can install the secrets directly on the device through the uNabto provisioning module - see the apps/weather_station demo for an example of using this module. Contact Nabto with respect to optional hosting of the central provisioning services necessary as part of this setup.

The Nabto provisioning solution supports different device secret installation methods, common for them all is that a secret can only be issued once for a device. If a secret needs to be re-installed, an administrator must actively allow the device to be opened for re-provision.

The uNabto device may specify nothing at all when connecting to the provisioning service - the service then assigns an arbitrary id and a device secret to the device.

The uNabto device may specify a full device id, the provisioning service assigns a device secret which may then be derived or otherwise mapped to this device id.

The uNabto device may specify a mac address, the provisioning service then derives a device id and associated key.

Finally, the uNabto device may combine any of the above with an additional *token* - a random string that grants permission to invoke the provisioning services. This prevents denial-of-service attacks where an attacker may provision specific device ids or mac addresses, preventing actual users from later provisioning their devices (as devices can only be provisioned once). The random string may then e.g. be printed on a piece of paper and put inside the product box for the user to enter and then passed on to the device through a Nabto RPC function. Or variants of this - e.g. the string can be encoded in a QR code for the user to scan. Or the token can be obtained by the user through a user management portal service if such is part of the solution.

9.2.2 Device based provisioning at the factory

To prevent the attacks described in the preceding section and simplify the end-user experience by eliminating the need for a user-entered token, the device may be provisioned at the factory: At a first boot, e.g. just before QA, the device contacts the provisioning service and has the id and key installed. Either a local provisioning service can then be used. Or a local proxy for the Nabto hosted service may then be used that holds the provisioning service API key to not expose this in the firmwares.

With this approach you get the best of both worlds - the simple manufacturing process with identical firmwares and no hassle for the users, the device works immediately when installed. At the price of an increased post-manufacturing procedure with the added provisioning step.

9.2.3 Client based installation at the end-user

With this approach, the user somehow obtains the device id and and secret. This is similar to obtaining the token described above, e.g. through a QR code provided on a piece of paper in the product box, scanned by the user at install time. Or it could be obtained through a central service, especially relevant if a user management portal is part of the solution. Once the id and secret pair is obtained, the user transfers the information to the target device on the local network through some means - e.g. through bluetooth or a Nabto RPC method.

Important note: Using Nabto RPC for such bootstrapping means transmitting the sensitive device secret in clear text on the local network. The token described in the previous section is not as sensitive as it is just used to ensure one-time invoking of the provisioning service; an eavesdropper has no use of that token after obtaining it (except doing a simple denial of service attack, if timing is very lucky and the owner is prevented from invoking the service).That is not the case when passing on the actual device secret on the local network - an eavesdropper may obtain the actual device secret and abuse it afterwards.

So the Nabto RPC method should *only* be used when users truly trust their local network - and/or for non-critical use cases, e.g. for monitoring statistics of a weather station. The local pre-shared secret based secure access as outlined in section 9.5 below is often not a practical solution in this scenario as it also involves pre-installing a secret - but the approach may be used for *obscuring* the transmission of the device secret on the local network instead of sending it in cleartext if using a common key. Still not secure, though, so all the above caveats still apply - although making it a bit more complicated to obtain the secret for the eavesdropper than a simple packet dump.

9.3 Factory Reset and Device Recycling

The device vendor must make sure the crypto key survives a factory reset by writing it to some persistent area so the device is not made unavailable after a factory reset at the user.

On the other hand, the vendor must also be very careful about how to design the device recycling process (e.g. how to handle a returned product): Before recycling the device, all access control lists should be cleared to make sure the previous owner does not have access to the device. That is, either the Nabto level access control list or the application level credential database must be cleared. Hence the new owner of the device will perform a new initial pairing and the previous owner cannot have access.

It is recommended that a factory reset performs this clearing of the access control list to make it easy for the user to make sure no one has access to the device.

For the highest level of security, the vendor will have to re-install the firmware on the device to make sure it is not tampered with (not different than any other electronics product that can be recycled). Additionally, for maximum security, a new unique Nabto device id and key (or token for delayed key installation) should be installed to prevent device impersonation if the previous owner extracted the device id and key from the firmware.

9.4 Roadmap feature: Central Custom Authorization

It is in the Nabto roadmap to enable custom, centralized authorization, simplifiying the management of access to devices. Prioritization of these additions and exact schedule is based on customer requests.

With the intended approach, custom functionality can be injected into to the basic Nabto connect handshake – for instance, once the user has been authenticated, custom logic can decide if the connect request should be passed on to the device: Is the user id black listed? Or geo-based decisions can be made from the user’s IP address – to allow e.g. only users in certain countries to connect to specific devices. Or an existing infrastructure can be used to query for permissions for specific users to access specific devices (e.g., an Active Directory service).

Additionally it is being considered to add facilities for central device privilege assignment: Instead of (or in addition to) maintaining privilege mappings on the individual device, privileges are assigned by custom code on the basestation and securely passed on to the device as part of the handshake. For instance, central custom logic can define that the current user is allowed to perform some specific operations on the device. This set of operations is communicated to the device that makes sure to later only execute requests from the user that adhere to the restrictions.

The privilege assignment could also be expressed as roles to minimize the coupling between the custom code on the basestation and the device code. That is, instead of defining specific privileges, the basestation can map the user to a specific role and pass this on to the device as part of the handshake. For instance, a user can be mapped to the role "Administrator" by the basestation and the device then allows certain operations accordingly.

The central privilege assignment can also be looked up in an existing directory infrastructure (e.g., Active Directory) or using a dedicated access management such as RSA Access Management.

Please contact Nabto for further details about this concept and let us know if you critically need this feature for a project.

9.5 Local Security

With Nabto Client SDK 4.2 and uNabto SDK 4.3 it is possible to have encrypted local communication using a local pre-shared key approach: A secret is injected into the uNabto SDK - it is up to the device vendor to decide and implement the exact details (perhaps the secret is installed at the factory, like the other secret for remote access described in section 9.2 above).

The same initial secret is somehow made available to the client application, again the details are up to the vendor - perhaps the secret is scanned from a QR code provided with the device. Once this initial connection has been established, the client can install a per client specific secret into the device.

For more information on using the local shared secret, see the documentation of nabtoSetLocalConnectionPsk in the Nabto Client SDK header file. And the uNabto SDK documentation of unabto_local_psk_connection_get_key callback function in unabto_app.h. To manage pre-installed keys in uNabto applications, the vendor can use the set-psk function in the uNabto ACL editing tool in unabto/apps/aclcli.

10 Export/Import Restrictions in US and France

The core Nabto platform uses cryptographic functionality from the OpenSSL libraries. When apps are distributed from the various app stores (Apple App Store and Google Play), this takes place from the US. Hence, US export restrictions on cryptographic technology apply and certain regulations must be adhered to. Similarly, France requires cryptographic apps to be approved prior to import into France from the US based app stores.

If your application will just use one of the standard Nabto apps available from the app stores, there are no problems related to the restrictions: Nabto's apps have already been approved for export by the US authorities and for import by the French authorities and can be used by your users without further worry.

However, if you want to create a custom app based on the Nabto Client API library or if you want to distribute an OEM version of one of the default Nabto apps, you must apply for approval at the relevant authorities. Note that an OEM version of the existing apps only requires approval if you want to distribute through your own iTunes or Google Play account – if your custom app is distributed with Nabto as vendor, Nabto’s existing approval is sufficient and we will take care of the paperwork when submitting the app.

For submitting Nabto based apps to Apple's app store, please see the Apple Developer guide on "Cryptography and US Export Compliance" (http://goo.gl/UBlbQV). The documentation and approvals described there also applies to apps distributed through Google Play.

As of 2016, the rules seem to have been relaxed a bit for US export. According to https://help.apple.com/itunes-connect/developer/#/devc3f64248f, it appears to no longer be necessary to apply for approval if your app uses standard algorithms (Nabto uses standard algorithms) and you no longer need to supply the ERN document.

11 Appendix A

11.1 Secure Device Registration with Basestation

The device registers with the basestation using a three way shared secret challenge response handshake. Two services are involved on the basestation - the "Controller" service that acts as a load balancer and the "GSP" service that acts as a registry of devices.

The handshake consists of five packets (see Figure 1 Packet Flow when Attaching to the GSP below). The first two packets are between the device and the controller. The device asks the controller to get the IP address of the GSP it should use.

The next three packets between the device and the GSP is the three way challenge response handshake. During the handshake a session key is created, this session key is used for the future communication between the device and the GSP. This communication is secured by an encrypt-then-MAC scheme, based on the algorithms AES-128 and HMAC(SHA-256).

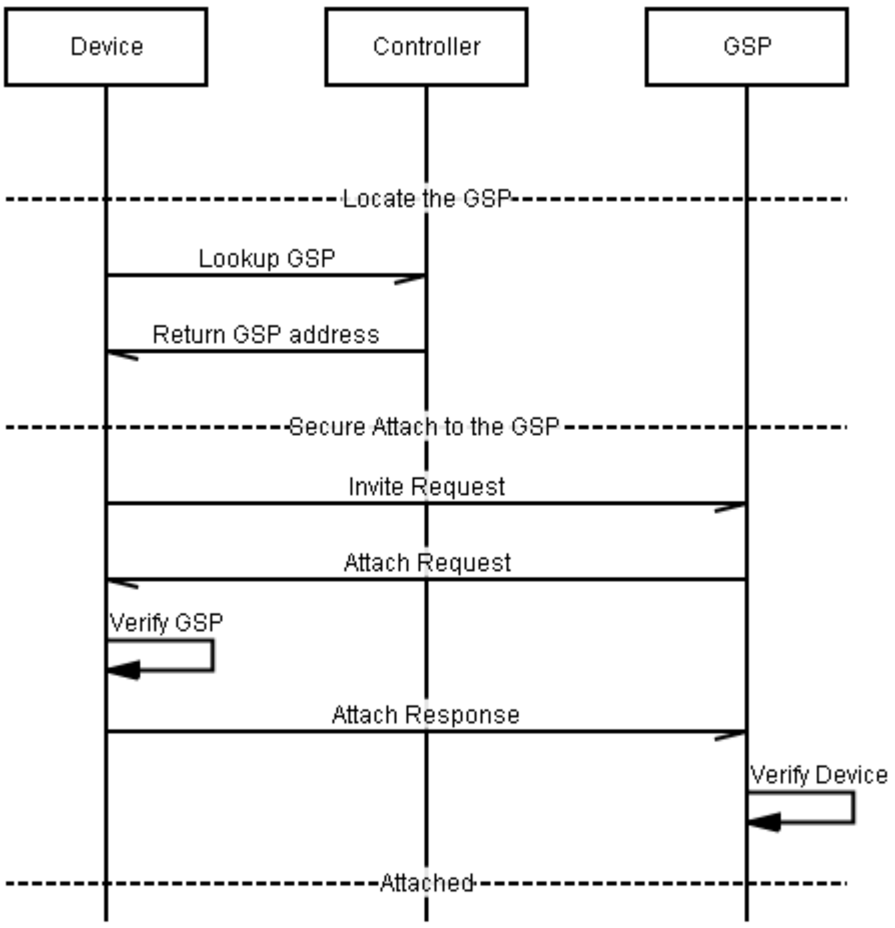


Figure 1 Packet Flow when Attaching to the GSP

11.2 Secure Client Connections to a Device

The connect sequence is a combination of a normal SSL handshake and communication with a device through a secure authenticated and encrypted channel (Figure 2). The reason for doing it this way is to offload the computationally heavy certificate verification to a trusted third party.

Between the Client and the GSP a normal SSL handshake occurs where each party is authenticated and a session key is calculated. The GSP sends the session key to the device using the secure channel described in section 11.1, including the certificate id of the client which the GSP has verified.

After the connect sequence both the Client and the Device knows the same session key, they will use this as a symmetric session key on the future communication on P2P or relay channels they will establish with each other. The channels are encrypted and authenticated by an encrypt-then-MAC scheme based on AES-128 and HMAC(SHA-256).

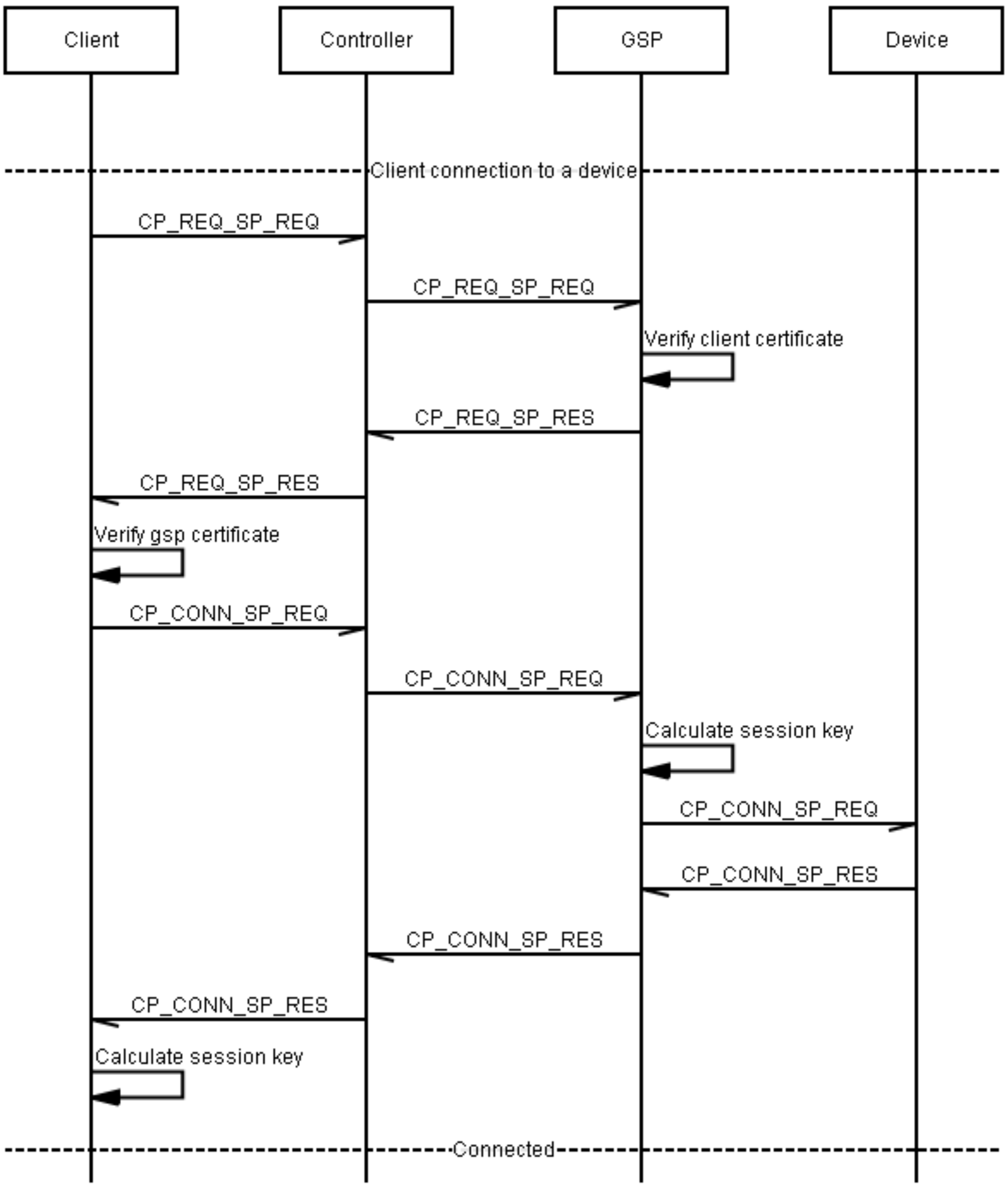


Figure 2 Connect packet sequence

11.3 Encryption and authentication functions in uNabto

In uNabto the function used to encrypt data is AES128-CBC and HMAC-SHA256 is used for message authentication codes. AES128-CBC works by chaining blocks together and encrypt them. AES128-CBC uses an

Initialization Vector hvis is sent together with the encrypted message the generation of the initialization vector should obey the following two constraints:

1. The initialization vector must be unique under a given key.
2. The initialization vector must be unguessable for an adversary at encryption time.

Requirement 1 is required such that the same message can be sent multiple time but still be indistinguishable for an adversary.

Requirement 2 is a bit more subtle but it boils down to an attack on the encryption if an adversary can guess the initialization vector and be able to make the peer encrypt a message the adversary chooses. See SSL 2.0, TLS 1.0 BEAST attack. There is not yet a known method for making a practical attack on the unabto protocol if the initialization vector is guessable at encryption time. But it cannot be denied that such method could exists, so the recommendation is to use an adversary ungessable initialization vector at encryption time.

11.4 Random Functions in uNabto

The random function in uNabto is used in the following places.

1. As initialization vector in AES128-CBC encryptions.
2. As a Nonce and as a Seed when creating a connection between the GSP and the device.

For AES128-CBC initialization vectors the requirement is that the random is unique and unguessable at encryption time for an adversary. This means that the random needs to come from a PRNG. If the platform comes with a fast PRNG then that can be used else the FORTUNA prng from libtomcrypt can be used. If the platform doesn’t come with a fast random function then a simple approach is to AES encrypt a 128bit counter where the counter and aes key is seeded with 256bit random at startup. This can then be reseeded if needed.