Security in Nabto Solutions

NABTO/001/TEN/036

1 Contents

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 11**. 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.

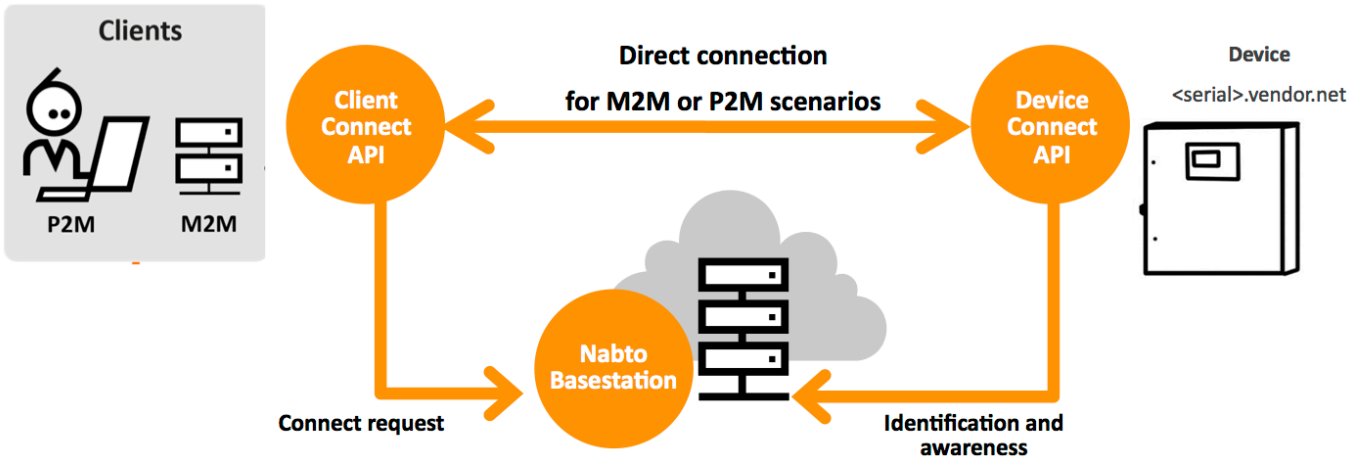
4 Legal Disclaimer

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 11.

5 Bibliography

TEN023	NABTO/001/TEN/023: uNabto SDK - Writing a uNabto device application
TEN024	NABTO/001/TEN/024: uNabto SDK - Writing a uNabto HTML client application
TEN025	NABTO/001/TEN/025: uNabto SDK - Writing a Nabto API client application

6 Nabto Platform Basics



The Nabto platform consists of 3 components:

- Nabto **client**: Binaries supplied by Nabto, used by the customer's HTML or native 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. Also supplies the user interface to Nabto HTML clients.

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 ("attach") when online and where clients look for available devices. After connection establishment, 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].

The customer's client application may use the Nabto client in different ways: The customer application can be an HTML application that uses the Nabto client to retrieve JSON data in a web application – in this scenario the Nabto client is typically a web browser plugin or mobile app, hosting the customer application. The latter is distributed from the basestation to the client and is denoted an HTML device driver bundle. Writing such an application is the topic of [TEN024].

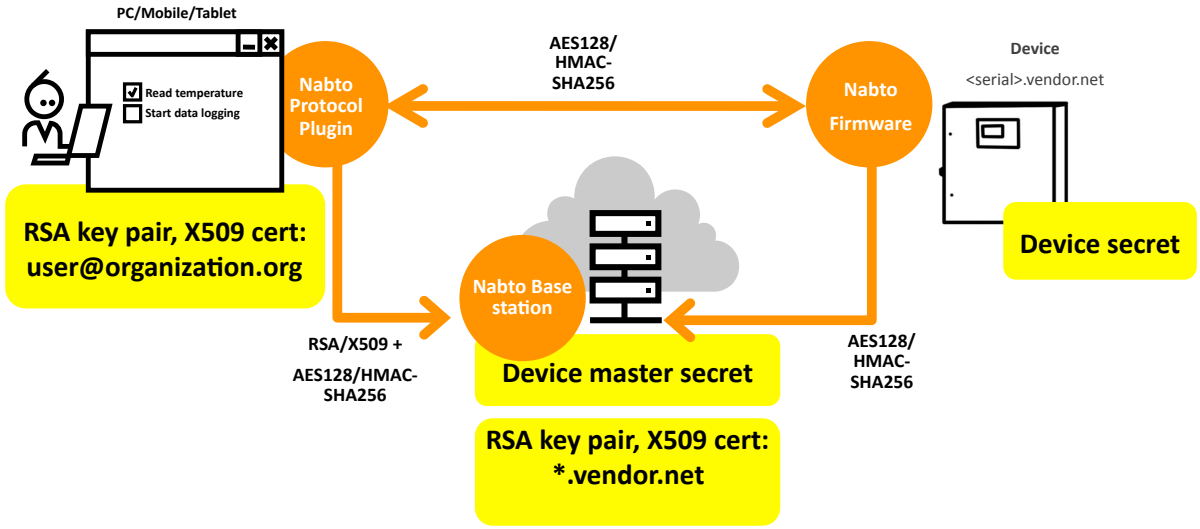
The customer's client application can also be a native (non-HTML) application, linked with a Nabto client API library. The native client application can use the same request/response mechanism to invoke the device as HTML applications do. Additionally, the native client can establish streaming data connections with the device – this is a popular way of adding seamless, secure remote access capabilities to legacy client and device applications.

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 basestation) and establish secure connection to basestation. Public key signed by CA of choice (e.g., Nabto CA), X.509 subject contains user's email address. Section 8 describes how to obtain client keys and certificates.
- **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 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.



The 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) full 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.

With the current platform, communication between two peers on the same local network is not encrypted. When full PKI support is available, local connections will also be encrypted.

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

8 Issuing Client Certificates

A basic premise in the Nabto communication platform is the RSA/X509 client key and certificate. It is used to authenticate the client towards the basestation. The authenticated identity is passed on to the device which in turn decides if the user is allowed to proceed and which operations can be performed (see section 10 for details on access control (authorization)).

Hence, a unique Nabto client certificate must be issued and signed for each user to be able to use the platform's features for authentication and authorization[1]. Nabto provides the full infrastructure for issuing such certificates at runtime.

If you just want a quick walkthrough of the steps required to use the platform for issuing certificates, please see the summary in section 8.3.

8.1 Identity Validation

To be able to issue a certificate, the user's identity must first be validated. Nabto provides an email based mechanism for this, similar to what is provided by many other services to validate that a user really has access to a given email account: An email is sent to an address provided by the user with a link to a Nabto webservice, specifiying a unique token. When the user clicks the link, a record is created in the central user database with the specified email address and user chosen password.

This database of validated user identities is then later queried when actually issuing the certificate.

To send the email with the token, the user either clicks the "Sign Up" button in the Nabto apps or plugins when prompted. Or in custom applications, the **nabtoSignup()** function is invoked – see TEN025, section "User Profile Management" for details on invocation and how to configure the webservices host to use in the clients.

8.1.1 Customization of Email Validation Flow

If just using the default Nabto apps and plugins or if not changing the configuration in API clients, the default webservice host will be used. Hence, users will end up in the standard Nabto user database and the verification mails will be sent from Nabto. If using your own OEM version of the Nabto apps or developing an API client, you can customize the emails and you control where user data are kept.

The email templates to customize are located in ~nabto/templates/MAIL on the webservice host if using the default Nabto webservice installation package. Email sender account and database configuration can be configured in ~nabto/lib/class.Configuration.php.

[1] In some situations it is desirable to bypass Nabto's features and rely on e.g. an existing application level implementation. In this situation a Nabto client certificate is not needed – Nabto then just provides confidentiality and integrity, the existing application takes care of authenticating the user e.g. through HTTP or RTSP basic auth, i.e. by querying for a username/password that is validated against an existing database. This approach is described in section 10.3.

When setting up a custom host for issuing certificates, remember to install an HTTPS certificate to secure the certificate issuing process.

8.2 Issuing the Client Certificate

To issue a certificate, the Nabto client invokes the Nabto webservices (introduced in the previous section).

The user must provide a username and password when a certificate is issued. The Nabto client creates an RSA key pair and sends the certificate sign request (public key) to the Nabto webservices along with the user specified username and password. If the credentials can be validated in the user database, the Nabto webservice signs an X509 certificate based on the public key. The certificate contains the user's username in the X509 subject field. The user then installs this certificate in the local Nabto certificate store. The user's private key is encrypted with the same password as supplied to the Nabto webservices.

To trigger the certificate issue procedure, the user clicks the "Sign In" button in the Nabto apps or plugins when prompted – if no certificate is present locally, the webservices are invoked as described above. In custom applications, the **nabtoCreateProfile()** must be invoked (see TEN025, section 6.6 "User Profile Management" for details).

8.2.1 Customization of Identity Validation Method

The Nabto webservices do not necessarily have to use the default user database populated with email verified identies as described above. This is just the default approach to ensuring certificates are only issued to validated users.

If other approaches are preferred, this is indeed possible – for instance, instead of populating the database of validated users as described above and later querying it when issuing the certificate, an existing identity management (e.g. Active Directory or RSA Identity Management services), can be used to authenticate the user. This will require a simple adaptation of the default certificate issue services (contact Nabto support for details). The end result is the same: The user gets issued an RSA/X509 certificate for use in the communication with Nabto-enabled devices.

8.3 Summary of Typical Integration Tasks

8.3.1 Bypassing Nabto's Authentication Mechanism

If using an application level authentication mechanism (e.g., if tunneling to an existing web application on the target device), Nabto's certificate handling described below can be ignored and the special "guest" account can be used. This ensures integrity and confidentiality when communicating with the remote peer, but any required authentication must be implemented at the application level as outlined in section 10.3.

8.3.2 Customizing Certificate Webservices

If using the default Nabto apps or default plugins, the standard Nabto user database and email templates will be used and nothing needs to be (or can be) configured.

If using an OEM re-branded Nabto app or plugin or using a custom client built with the Nabto Client API, the Nabto webservices host to use in clients and webservice content templates can be customized.

8.3.3 Issuing a Client Certificate

The Nabto apps and plugins guides the user through the signup and certificate issuing process so nothing needs to be to done to ensure the process get triggered correctly.

Custom clients built with the Nabto Client API must use the Nabto user profile management functions described in TEN025 section 6.6.

9 Installing uNabto Device Crypto Keys

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.

Different models exist for the installing shared secrets: The conceptually simplest 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: 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. Hence, a more flexible post-factory based key installation approach is also supported by the framework as outlined in section 9.2 below.

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 advanced platform capabilities (HTTPS support) and 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. But the key could also be retrieved e.g. from a local uNabto client (who e.g. scanned a QR code) or entered by the user through a panel on the device.

9.1.1 Generating uNabto Device Crypto Keys for Test / Development

For evaluation and development, the demo basestation and management services available on https://portal.nabto.com can be used. It is possible to generate up to 10 device id/key pairs in the demo.nab.to domain.

9.1.2 Generating uNabto Device Crypto Keys for Production

On the basestation in the nabto user's home directory (~nabto/.nabto), the basestation's mastersecret and/or specific keys for individual devices are installed in the secrets.json file:

```
{
    "version": 1,
    "regex_match": [ {
        "matches": ".*\\.p2p\\.vendor\\.net",
        "mastersecret":
          "49e7c009a2795a635e98936c241e80746bf0a44e28f8009cc2a1c3eba1b855e4"
    } ],
    "exact_match": [ {
        "matches": "x1.p2p.vendor.net", "presharedkey": "33b9fa6d52a895fc081e14424cdc8ac0",
        "matches": "x2.p2p.vendor.net", "presharedkey": "075f2d95209bd8b846d1d43edaac332a",
        "matches": "x3.p2p.vendor.net", "presharedkey": "3b16f43f641b9de8bc02b04925e3fc11"
    }]
}
```

Based on this mastersecret and the individual devices' ids, shared secrets are derived using HMAC/SHA256. At runtime, the basestation performs this derival to establish a secure connection to devices. Or preshared keys (not derived) can be specified as seen.

For generating device keys, derived from the mastersecret, to be installed on the individual devices, tools are installed on the basestation (~nabto/bin/unabto-shared-secret.pl).

It is simple to build custom key generation services as most platforms provide easy-to-use HMAC implementations. For instance, many manufacturers prefer to invoke the key generation as a webservice from within the manufacturing process - in PHP it can be done as follows:

```
function generateSecret($secret_key, $device_id) {
  echo substr(hash_hmac("sha256", $device_id, pack("H*", $secret_key)), 0, 32);
}
```

9.2 Key Generation at Runtime

Instead of installing a permanent device key on the factory, the key can be downloaded by the device at runtime from a central key generation service. Two approaches can be used here: Either a secure token based approach where the device must present a valid token to get issued a shared secret (described in section 9.2.1 below). Or a less secure but more convenient WPS-like approach where the device is allowed to download a new key in a certain time window - without further authentication (described in section 9.2.2).

9.2.1 Generate Key Using Token-Based Authentication

With this approach, the device must present a key-generation token to download a new key - an initial such token is installed on the factory. The manufacturer then does not need to handle a full list of sensitive device keys but only the tokens that allow access to the key generation services where in turn access can be carefully monitored and tokens discarded.

The key generation service is invoked by the Nabto SDK on the device in a manner corresponding to the following:

```
$ curl -XPOST https://nabto.vendor.net/api/1/devices/e09ca8.vendor.net/\
       shared_key?token=9b0d2ebee488073bb918e4715b03f289
{
  shared_key: "8f83520f554ea6c9e9393ba72d2a4153"
}
```

At the first boot, the device uses the factory token to securely download a device key using HTTPS towards the key generation service. The manufacturer can choose to perform this initial boot, e.g. as part of a QA procedure, or it can take place at the consumer.

Nabto provides a token generation service that can be used from the factory and custom vendor applications. For instance, the vendor has a user portal where key generation tokens can be purchased by the end-user (e.g. denoted "activation keys" or similar):

```
$ curl -XPOST https://nabto.vendor.net/api/1/devices/e09ca8.vendor.net/token
{
  token: "9b0d2ebee488073bb918e4715b03f289"
}
```

The device may now create a shared secret by passing this token to the key generation service.

Access to this token generation service is restricted either through the network (local net or VPN) or through HTTPS basic auth to make sure only the vendor can invoke the service from the vendor's user portal or customer service system.

The user portal invokes this Nabto token generation service and passes the token back to the user who in turn passes it on to the device. If on the same local area network, the token can be passed directly to the device - otherwise, the vendor must consider ways to do this (e.g. on camera products, the device scans a QR code displayed on the user's phone, representing the token).

9.2.2 Generate Key Using WPS Like Provisioning

A simpler but less secure alternative to the token based approach is to use WPS like provisioning: Instead of passing a token from a central application to the device to allow it to retrieve a key, the user may mark the device for being in "provisioning mode" (again, using central services as provided by Nabto):

```
$ curl -XPOST https://nabto.vendor.net/api/1/devices/e09ca8.vendor.net/provision_window
{
  provisioning_window_ends: "1420427043"
}
```

Once a provisioning window has been created, the vendor's app triggers the Nabto framework on the device to activate key download on the device (e.g. by rebooting and pushing a button or explicitly using local communication). The device presents its id to the central services that see this device as ready for provisioning and a new key may be downloaded.

9.3 Factory Reset and Device Crypto Keys

The device vendor must either make sure the crypto key survives a factory reset by writing it to some persistent area. Or guide the user through token generation to enable the device to generate a new key after a reset as outlined in section 9.2.

10 Application Level Security

Sections 8 and 9 above described how to configure and prepare the builtin security capabilities of the Nabto framework to support basic security features in the platform. This section describes how to use the framework from within applications to ensure communication and interaction is secure.

10.1 Secure Sessions in uNabto HTML Applications

Nabto HTML applications are vulnerable to cross-site attacks and precautions must be made by the application developer to prevent this. Consider an attacker that either serves the following snippet directly from his own website or perhaps manages to inject it into a popular, regular http based website:

```
<img src="nabto://2d88ef44.p2p.vendor.net/shut_down.json?"/>
```

Without the user knowing about it, the user sends a shutdown request to the indicated device by visiting a completely different website. The Nabto framework supports preventing this by requiring all requests to supply a session key, in turn supplied to the HTML application after the user logs in.

To enable session keys, add a manifest.json file to the root of the Nabto HTML Device Driver bundle (see TEN024 "HTML-DD content" for an intro to the bundle structure) with a `sessionKey` field:

```
manifest.json
{
  "manifest_version": 1,
  "require_session_key": true
}
```

If trying to access the shut_down URL now as above, the request is not executed and an error is returned to the user:

```
{
    "error": {
        "event": "2000059",
        "header": "Session key missing (2000059)",
        "body": "The session key is not present, and the security configuration for the html
device driver requires a valid session key"
    }
}
```

When the Nabto client serves the first page in the HTML DD bundle, it passes along the session key as a unique string. Whenever a request is to be executed subsequently, the caller must supply the same key. Hence, the HTML application must store the session key or make sure to propagate it amongst pages:

```
function shutdown() {
  jNabto.request("/shut_down.json?session_key=" + sessionStorage["session_key"], ...});
}
```

10.2 Implementing Nabto Level Access Control on Device

Access control (authorization) is implemented at the uNabto application level by using the primitives provided by the framework to access the user's identity and/or type of connection to the device. Implementation details are provided in TEN023 section “Access Control”.

The uNabto framework provides the uNabto application with the authenticated identity of the user when a connection is established: The basestation has validated the user’s X509 certificate and extracted the user id which is passed on to the device on the encrypted channel between device and basestation as part of the connection request.

Additionally, whenever a uNabto application event callback is fired (that is, when the device handles an individual uNabto web server request), the event is associated with the identity of the active user.

The connection type is available to the applicaction to tell how the user is connected (local vs remote - the user is only authenticated on remote connections). Typically, the device's ACL configuration is empty when delivered from factory and an administrator has to setup privileges on a local connection.

Based on these simple operations, access control can be implemented by maintaining access control lists and privilege mappings. ACL and privilege mappings can be maintained through the uNabto web interface through

special application events (for instance, add user to ACL, remove user etc.). Nabto provides an application level module for maintaining the access control lists to simplify such implementation, see `unabto/src/modules/acl` in the uNabto SDK.

Maintaining access control lists can be done from a central location by running a Nabto Client API application that pushes ACL updates to the individual device, for instance based on some central configuration. Also, see section 10.5 below about an upcoming feature for centralizing access control.

10.3 Authentication at the Application Level

10.3.1 HTTP/RTSP Basic Authentication

Applications may implement their own means of authentication, bypassing Nabto's built-in mechanisms. This is typically used in tunneling/streaming applications where an existing HTTP or RTSP based application is accessed through a Nabto tunnel where 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. Communication between client and basestation/device is still encrypted and is hence confidential and with integrity checking with a secure channel established to the device based on its shared secret. But client authentication is performed by the existing application by verifying a user specified username / password combination.

10.3.2 Automatically Generated Credentials

Instead of letting the user enter credentials for accessing a device (e.g. through HTTP/RTSP basic authentication as described above), the client app vendor can hide this complexity: Hence, instead of prompting the user for username and password, as part of the initial provisioning, the application generates credentials unique for the client in question. These credentials are then passed on to the device for instance on a local connection and added to a local database on the device. Essentially, the specific *client application instance* is now paired with the device - the *user* as such is not paired with the device.

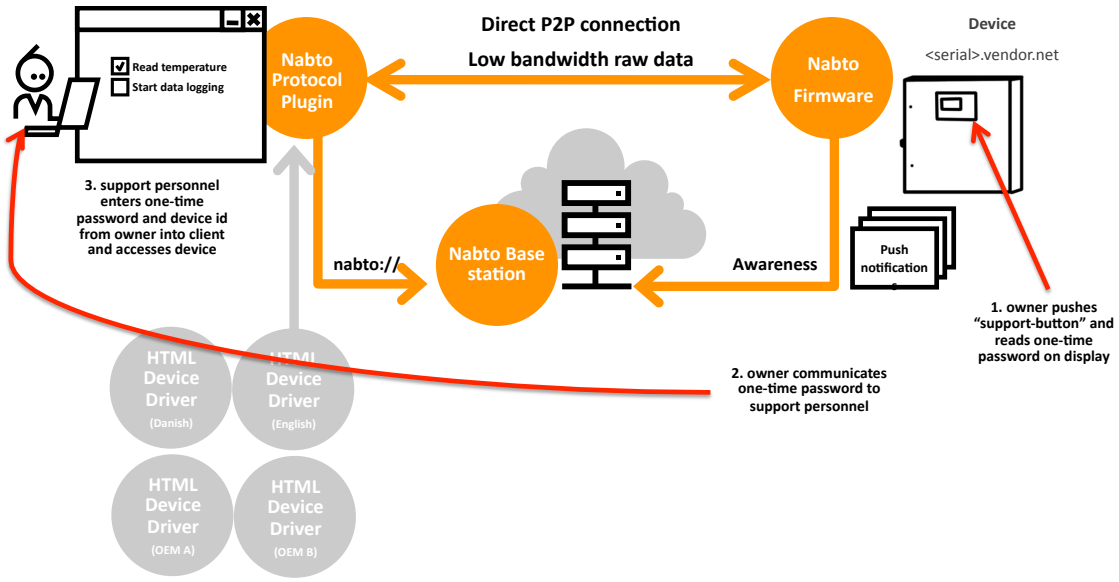
If for instance a user then wants to share access to a device, the automatically generated credentials can then be transferred to another user or new credentials can be created for that purpose. The same if the user wants to access the device from another client application - the credentials must somehow be transferred to this alternative client. This can be done in many ways - simply reading the auto-generated string on the initial client and entering them on the new one. Or more sophisticated approaches like scanning of QR codes or Apple's AirDrop can be used for sharing the credentials.

10.4 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 uNabto web requests: 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 is sketched in the drawing below: The device is 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.



10.5 Custom Authorization on Basestation

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.

10.6 Device Recycling and Factory Reset

The vendor must 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.

11 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.

12 Appendix A

12.1 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).

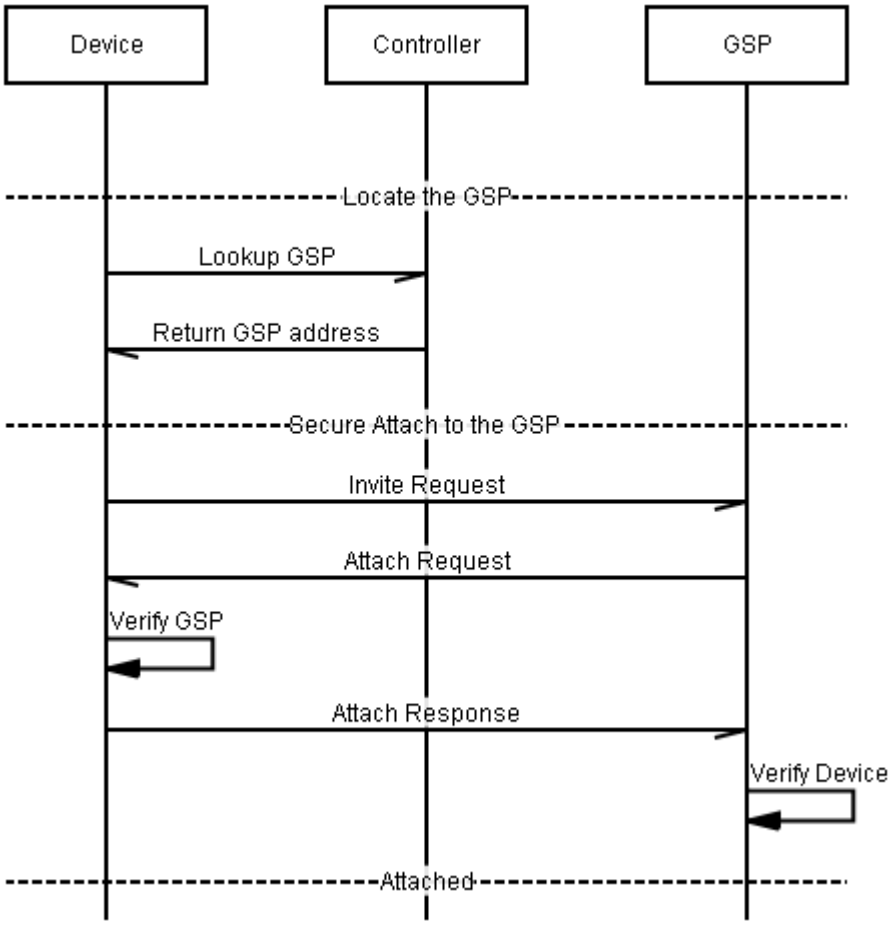


Figure 1 Packet Flow when Attaching to the GSP

12.2 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 12.1, 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).

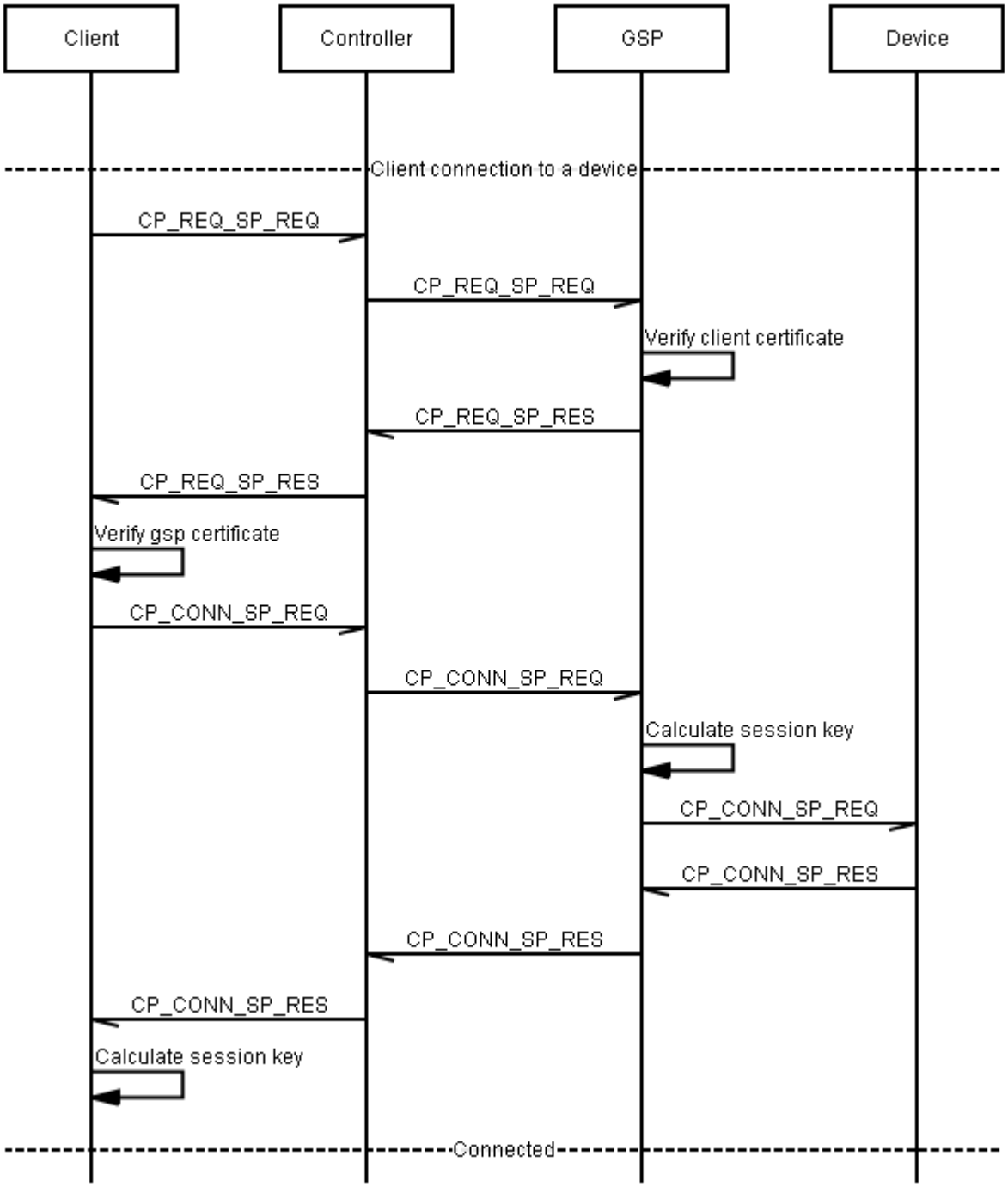


Figure 2 Connect packet sequence