

Authentication and Authorization

Thomas Braun

byte physics e. K.

21th October 2026

Welcome to day 2!

Recap of day 1

We now have an encryption proposal for Tango which ensures that all communication is encrypted and we know who we are talking to. And which is (mostly) PQC-ready.

Basics I/II

- ▶ Authentication is the act of proving an assertion, such as the identity of a computer system user. In contrast with identification, the act of indicating a person or thing's identity, authentication is the process of verifying that identity.

Basics II/II

- ▶ Authorization in information security (...) is the function of specifying rights/privileges for accessing resources, in most cases through an access policy, and then deciding whether a particular subject has privilege to access a particular resource.

Foundational principles

- ▶ Secure by default
- ▶ Centralized configuration without single point-of-failure
- ▶ Scalable
- ▶ Robust

Basic idea

- ▶ Write a new DS, the AuthDS, which offers a command to authenticate users/machines
- ▶ The AuthDS has a table with the access rights for each user
- ▶ This new command returns a digital ID card which encodes the access rights
- ▶ The client attaches the ID card to its Request-Reply calls
- ▶ cppTango then checks if the attached ID card allows the requested action
- ▶ The TDB holds the list of trusted AuthDS public keys (the ID card issuers) and is thus the trust anchor for authorization

In the following we
elaborate on the
details

AuthDS details I/IV

- ▶ C++17 DS using POGO
- ▶ Authorization requires that cppTango was compiled with encryption support
- ▶ Refuses to start with an encryption mode other than 'required', so only encrypted connections are possible

AuthDS details II/IV

- ▶ Needs to run as device name `sys/auth/XXX`
- ▶ One or multiple instances, all with the same backend data
- ▶ The DS will support authentication via user/password (A), with optional MFA support, LDAP/OAuth (B) or access tokens, also known as API keys (C). The latter are envisioned for automated workflows and the client part of device servers.
- ▶ Has an additional pair of public/private keys per instance

AuthDS details III/IV

- ▶ Uses authorization like every other device server, for its own device and its admin device. The only exception is the `Auth` command, which is callable without authorization.
- ▶ Multi-step authentication (MFA, OAuth) is always completed on the same AuthDS instance.
- ▶ Rate-limits the `Auth` calls per client certificate before computing any password hash

Dependencies

AuthDS:

- ▶ OpenSSL (everything crypto including Argon2id, already via cppTango)
- ▶ curl (OAuth, i.e. SSO via Keycloak)
- ▶ libfido2 (FIDO2)
- ▶ libldap2 (LDAP)
- ▶ rapidyaml (configuration file parsing)
- ▶ OATH Toolkit (TOTP)
- ▶ jwt-cpp and nlohmann::json (JWT)

cppTango:

- ▶ jwt-cpp and nlohmann::json (JWT)

Command interface

```
1 using StrArr = DevVarStringArray;  
2 StrArr Auth(StrArr creds);
```

A cppTango CLI application will be provided to showcase MFA usage.

Auth example calls

Input:

- Str[0] "User"
- Str[1] userA
- Str[2] "Password"
- Str[3] superSecretPassword
- Str[4] "Device"
- Str[5] a/b/c
- Str[6] "Attr" or "Cmd" or ... [optional]
- Str[7] "my_attr" [optional]

Return (without MFA) or exception:

- Str[0] JWT

Auth: MFA/OAuth

- ▶ Similar to the single factor case above, just with a little more back and forth

End of the
presentation for this
session

AuthDS: Configuration file format

- ▶ The configuration is stored in a YAML file
- ▶ The AuthDS only reads the YAML file and never writes to it
- ▶ As the configuration file can contain TOTP seeds, it should be treated as a secret.
- ▶ A schema will be provided


```
23     - device: "e/f/g"
24       commands:
25         - name: "cmd0"
26           perm: "x"
27 - name: "admin"
28   permissions:
29     - device: "*/*/*"
30     attributes:
31       - name: "*"
32         perm: "rw"
33     commands:
34       - name: "*"
35         perm: "x"
36
37 users:
38   - name: "my_user"
39     group: "group_a"  # a user can only be part of one group
40     auth: internal  # alternatives are ldap or oauth
41                  # alternatives are none or FIDO2, the additional
42                  # config entries these need are not shown
```

```
43 mfa: TOTP
44 # only present when auth is internal
45 # Argon2id hash for the password "secret"
46 password_hash: "$argon2id$v=19$m=65536,t=3,p=4$jkkgWzr..."
47 # optional, for the example token
48 # tgat_7f3k9q2a_Xb8mK2pQvR4tY7wZ1nC5hJ9dF3sL6gA0eUeDpCC9cmF
49 token:
50   - id: 7f3k9q2a
51     secret_hash: 50a055958df7ea2df18cbe791ed5dde7ee96e059bbf
52     # UTC timestamp,
53     # exactly in the form YYYY-MM-DDTHH:MM:SSZ
54     expiration_date: "2027-10-02T16:32:26Z"
```

Access token

- ▶ Following good practice, we also support access tokens for automated workflows
- ▶ The token looks something like `tgat_7f3k9q2a_Xb..` where `tgat` is the prefix, `7f3k9q2a` the token id and `Xb...` the secret
- ▶ The AuthDS stores the id and the hashed secret
- ▶ `cppTango DeviceProxy/AttributeProxy` objects automatically read the environment variables `TANGO_AUTH_CLIENT_TOKEN` and `TANGO_AUTH_CLIENT_TOKEN_PATH` to get authentication credentials. This allows using authentication with zero code changes.

Environment variables I/II

- ▶ `TANGO_AUTH_MODE`: Set the authorization mode, valid values are required and off, unknown values result in a fatal error. This allows using authorization with zero code changes.
- ▶ `TANGO_AUTH_CLIENT_TOKEN`: See below
- ▶ `TANGO_AUTH_CLIENT_TOKEN_PATH`: Absolute path to a file with a token. The file is re-read when the JWT is about to expire, which allows updating the access token without a DS restart. Setting both `TANGO_AUTH_CLIENT_TOKEN` and `TANGO_AUTH_CLIENT_TOKEN_PATH` is a fatal error.

Environment variables II/II

- ▶ `TANGO_AUTHDS_PRIVATE_KEY`: Private key used to sign JWTs
- ▶ `TANGO_AUTHDS_PASSWORD`: Needed in case `TANGO_AUTHDS_PRIVATE_KEY` is password-protected
- ▶ `TANGO_AUTHDS_CONFIG_PATH/`
`TANGO_AUTHDS_CONFIG_INLINE`: The configuration file location or its contents

The order of precedence (lowest to highest) for the authorization mode:

- ▶ Function call
- ▶ Environment variable
- ▶ CLI flag

Permissions per operation I/II

- ▶ `read_attributes*`,
`read_attribute_history*` -> read attribute
- ▶ `write_attributes*`,
`set_attribute_config*` -> write attribute
- ▶ `write_read_attributes*` -> read+write attribute
- ▶ `command_inout*`, `command_inout_history*`
-> execute command
- ▶ `ZmqEventSubscriptionChange`,
`EventConfirmSubscription` (admin device)
-> read attribute, device access for interface change events
- ▶ CORBA commands and attributes -> device access

Permissions per operation II/II

- ▶ Other commands of admin device
dsriver/exec/instance -> execute (and maybe additional)
- ▶ Pipes -> denied
- ▶ Device access means that the JWT has an entry for the device, independent of entity and permission.
- ▶ Collocated calls (a DeviceProxy to a device in the same process) don't require authorization.
- ▶ The DS code is responsible for collocated calls it makes on behalf of a remote client, e.g. a client allowed to execute a command on a/b/c can trigger code which writes an attribute of e/f/g in the same process.

CORBA commands and attributes:

- ▶ ping, info*, black_box, command_list_query* (includes command config), command_query*, get_attribute_config*, IDL attributes: name, description, state, status, adm_name, ...

cppTango API additions I/VII

```
1  enum class Entity
2  {
3      Attribute = 1,
4      Command  = 2,
5      Device    = 3
6  };
7
8  enum class Permission
9  {
10     Read      = 1,
11     Write     = 2,
12     Command   = 3
13 };
14
15 // Server
16 enum class AuthorizationMode : unsigned int
17 {
18     Required = 1,
19     Off      = 2
20 };
```

cppTango API additions II/VII

```
20
21  // Server, needs to be called before Util::init()
22  static void Util::set_authorization_mode(AuthorizationMode auth)
23  static AuthorizationMode Util::get_authorization_mode();
24
25  class AuthenticationCredentials // Client
26  {
27      // Reads TANGO_AUTH_CLIENT_TOKEN or TANGO_AUTH_CLIENT_TOKEN_P
28      AuthenticationCredentials();
29
30      // Think about using something like SecureString from TDB for
31      explicit AuthenticationCredentials(std::string_view token);
32      explicit AuthenticationCredentials(std::string_view user,
33                                         std::string_view password)
34
35      void limit_permissions(std::string_view device);
36      void limit_permissions(std::string_view device,
37                             Entity ent,
38                             std::string_view entity,
39                             Permission perm);
```

cppTango API additions III/VII

```
40     // Contact the AuthDS to try getting a JWT
41     // Throws on errors, potentially returns information for MFA
42     std::optional<std::vector<std::string>> request();
43
44     // interface to support MFA
45     // Throws on errors
46     void request_mfa(std::vector<std::string> info);
47
48     // check if the authentication is valid
49     // invalid reasons can be something like failed MFA, unknown
50     bool is_valid() const;
51 };
52
53 class DeviceProxy
54 {
55     // throws if AuthenticationCredentials::is_valid returns false
56     // this avoids a network round trip as the device server would
57     // feature, just faster
58     void set_auth_creds(AuthenticationCredentials creds);
59 };
```

cppTango API additions IV/VII

```
60  class AttributeProxy
61  {
62      // ditto
63      void set_auth_creds(AuthenticationCredentials creds);
64  };
65
66  /// Internal APIs (client and server)
67  ///@{
68
69  class AuthenticationCredentialsImpl
70  {
71      // ...
72      JWT GetJWT();
73  };
74
75  class JWT
76  {
77      explicit JWT(std::string_view jwt);
```

cppTango API additions V/VII

```
78
79 // checks if JWT is well-formed, signed by a trusted AuthDS
80 // (currently in the cached TDB list) and
81 // the JWT itself has not expired
82 // (i.e. all checks which don't depend on the entity)
83 bool is_valid() const;
84
85 // check if the JWT allows access to the entity
86 // `ent` named `name` in mode `perm`
87 bool is_authorized(Entity ent,
88                    std::string_view name,
89                    Permission perm);
90
91 // the `jti` claim, for logging in device code
92 std::string_view jti() const;
93
94 ...
95 };
```

cppTango API additions VI/VII

```
96
97  /// Return the JWT of the current Request-Reply call, if any
98  /// Uses thread-local storage internally
99  std::optional<Tango::JWT> Util::get_jwt() const;
100
101  /// Functions to check if an operation is authorized,
102  /// XXX is the IDL version (most likely 8)
103  bool DeviceImpl_XXX::is_authorized(Entity ent,
104                                     std::string_view name,
105                                     Permission perm);
106
107  /// Throw if not authorized
108  void DeviceImpl_XXX::ensure_authorized(Entity ent,
109                                          std::string_view name,
110                                          Permission perm);
```

cppTango API additions VII/VII

```
111
112 /// Disable authorization on the given entity.
113 /// This is required by device servers acting as
114 /// generic services like the TDB/AuthDS only.
115 /// Using that is a rare exception and needs to happen inside
116 /// init_device.
117 void DeviceImpl_XXX::disable_authorization(Entity ent,
118                                             std::string_view name
```

End of the
presentation for this
session

JSON Web Tokens (JWT) I/IV

- ▶ Designed to encode claims between two parties
- ▶ Standardized as RFC 7519 (JWT) and 7515 (JWS)
- ▶ JWT is pronounced like the English word „jot“
- ▶ Consists of Header, Payload and Signature of payload
- ▶ Configurable lifetime, maximum 1d. As there is no JWT revocation, the lifetime is the delay until removing a user or a permission takes effect. Faster revocation is only possible by removing the AuthDS.

JSON Web Tokens (JWT) II/IV

- ▶ The header typ is tango+jwt.
- ▶ The device name in the tango claim acts as the audience, so there is no aud claim.
- ▶ The tango claim contains the patterns from the configuration, e.g. attr*, as the AuthDS does not know which attributes and commands a device has.

JSON Web Tokens (JWT) III/IV

- ▶ A JWT is bound to the client certificate it was issued for (fpXXX). The receiving device server compares it with the fingerprint of the TLS peer certificate of the current connection and rejects the request on a mismatch.
- ▶ The fingerprint is the hash XXX over the DER-encoded certificate, base64url-encoded without padding. The hash is exchangeable, but the verifier only accepts hash names from a fixed list, currently sha3-384 only.

JSON Web Tokens (JWT) IV/IV

- ▶ The `jti` is random. The AuthDS logs it together with the user or token id, so the device server logs, which only contain the `jti`, can be mapped to a user without revealing the user name to the device server.
- ▶ The verifier only accepts the algorithm matching the type of the AuthDS key from the TDB list, e.g. ES384 for a P-384 key.
- ▶ Encoded and signed, the example JWT has fewer than 1000 characters

JWT Header

```
1      {  
2          "alg": "ES384",  
3          "typ": "tango+jwt"  
4      }
```

JWT Payload

```
1  {
2    "ver": 1,
3    "iss": "sys/auth/3",
4    "fp#sha3-384": "sm9difDBXQ6vy73YYEUu0Ef...",
5    "iat": 1791561600,
6    "nbf": 1791561600,
7    "exp": 1791579600,
8    "jti": "Xb8mK2pQvR4t",
9    "tango": {
10      "a/b/c": {
11        "attrs": {
12          "attr*": "r",
13          "my_attr": "rw"
14        },
15        "commands": {
16          "cmd0": "x"
17        }
18      }
19    }
20  }
```

End of the
presentation for this
session

Chain of trust I/II

- ▶ How does a device server checking a JWT know that it was signed by a trusted entity? A new TDB command returns the list of trusted AuthDS.
- ▶ We also deliberately distinguish between X.509 certificates used for encryption and our AuthDS private/public key pairs used for authorization. This is done to ease future upgrades when X.509 certificates are replaced.

Chain of trust II/II

- ▶ It is acceptable that the JWT signatures are not PQC-safe: unlike encryption, signatures are not exposed to „store now, decrypt later“. BSI recommends classical signatures alone until the end of 2035, European Cybersecurity Certification Group admits them until at least 2033.
- ▶ JWTs are signed with ES384 (ECDSA with P-384 and SHA-384).
- ▶ While we recommend ES384, it is also possible to use EdDSA (Ed25519) or another supported asymmetric algorithm

New TDB command I/III

```
1  /// Return a list of devices and their public keys
2  ///
3  /// Does not require authentication but does require an
4  /// encrypted connection.
5  ///
6  /// Returns:
7  /// Str[0]: sys/auth/1
8  /// Str[1]: Public key (PEM format)
9  /// Str[2]: sys/auth/3
10 /// ...
11 Tango::DevVarStringArray GetAuthServerDetails();
```

New TDB command II/III

- ▶ This command will be called by cppTango every 5 to 10 minutes (randomly determined) to get a list of available AuthDS servers.
- ▶ A JWT whose iss is not in the cached list triggers an immediate refetch, at most once per minute, so a new AuthDS works at once.
- ▶ If the TDB can't be reached, cppTango keeps the last list and retries indefinitely.
- ▶ A removed AuthDS key therefore stays trusted until the next successful refresh.

New TDB command III/III

- ▶ The actual DS to be used is picked randomly from the available ones; if it can't be reached, the next one is tried. This list is not configurable via Tango but can only be edited outside Tango (admin user of DBMS behind TDB) for security reasons.
- ▶ Using an attribute was considered, but since (change) events can't be turned off on the server side, this is not safe, as it doesn't scale on the TDB level.

End of the
presentation for this
session

Finalizing discussion

- 3 Recap of day 1
- 4 Basics I/II
- 5 Basics II/II
- 6 Foundational principles
- 7 Basic idea
- 9 AuthDS details I/IV
- 10 AuthDS details II/IV
- 11 AuthDS details III/IV
- 12 Dependencies
- 13 Command interface
- 14 Auth example calls
- 15 Auth: MFA/OAuth
- 17 AuthDS: Configuration file format
- 21 Access token
- 22 Environment variables I/II

- 23 Environment variables II/II
- 24 Permissions per operation I/II
- 25 Permissions per operation II/II
- 27 cppTango API additions I/VII
- 28 cppTango API additions II/VII
- 29 cppTango API additions III/VII
- 30 cppTango API additions IV/VII
- 31 cppTango API additions V/VII
- 32 cppTango API additions VI/VII
- 33 cppTango API additions VII/VII
- 35 JSON Web Tokens (JWT) I/IV
- 36 JSON Web Tokens (JWT) II/IV
- 37 JSON Web Tokens (JWT) III/IV
- 38 JSON Web Tokens (JWT) IV/IV
- 39 JWT Header

- 40 JWT Payload
- 42 Chain of trust I/II
- 43 Chain of trust II/II
- 44 New TDB command I/III
- 45 New TDB command II/III
- 46 New TDB command III/III