

Encryption proposal

Thomas Braun

byte physics e. K.

20th October 2026

TB's SIG meeting plan

- ▶ 2 days
- ▶ One day for each topic
- ▶ 20th (today!): Encryption
- ▶ 21th (tomorrow): Authentication and Authorization
- ▶ 5 sessions on each day
- ▶ The first four sessions will be introduced by a short presentation by TB, and then we can discuss the ideas from it.
- ▶ The last session each day is reserved for finalizing the proposal

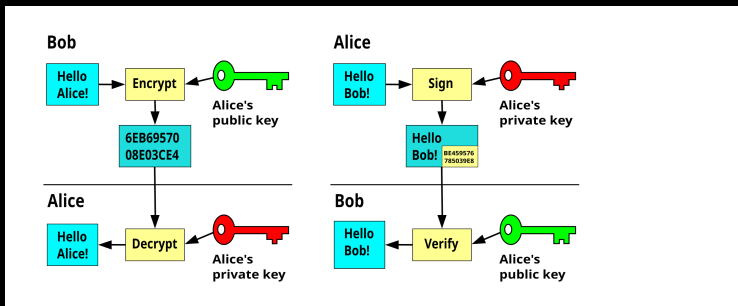
Why encryption for network messages?

- ▶ Confidentiality: Ensure that only allowed parties can see the message
- ▶ Integrity: Nobody has „fiddled“ with the message content
- ▶ Authentication: The sender of the message can be verified
- ▶ Non-repudiation: The sender of the message can later **not** claim that she was not the sender.¹

¹Offline example: Paying with credit card and PIN $\Rightarrow$ Your bank says that you can't repudiate the payment.

Common digital cryptography concepts

- ▶ Symmetric encryption: $\text{ALG}(\text{key}, \text{cleartext}) \Rightarrow \text{ciphertext}$ and $\text{ALG}(\text{key}, \text{ciphertext}) \Rightarrow \text{cleartext}$
- ▶ Public-key cryptography: Encryption and signatures



- ▶ Hash functions: $\text{HASH}(\text{data}) = \text{hash}$ where $\text{len}(\text{hash}) \ll \text{len}(\text{data})$ without collisions

Cryptography is always on the move I/II

Data Encryption Standard (DES) from 1977 with 56 bits of security: Encryption key was recovered after 56 hours by brute-force using custom designed chips in 1999 by the EFF². Not used in your favourite browser anymore and formally disallowed in 2011³. Even the extended version (Triple-DES) was disallowed in 2023.

²https://en.wikipedia.org/wiki/EFF_DES_cracker

³<https://www.rfc-editor.org/info/rfc6176/>

Cryptography is always on the move II/II

Secure Hash Function SHA-1: Introduced in 1995, mathematically broken since 2005, collision attack in 2017, phased out by browser vendors in the same year⁴, should be fully retired by 2030 according to NIST.⁵

⁴<https://blog.mozilla.org/security/2017/02/23/the-end-of-sha-1-on-the-public-web/>

⁵Still used in 2026 in fringe DVCS'e like git

Post Quantum Cryptography I/II

- ▶ 1994: Peter Shor released an quantum algorithm⁶ for finding prime factors of an integer. Which requires only polynomial and not exponential time.
- ▶ This „just“ needs a big enough quantum computer. Expected to land around 2040.
- ▶ This would make classical public-key encryption (RSA, finite-field/elliptic-curve Diffie-Hellman key exchange) insecure.

⁶https://en.wikipedia.org/wiki/Shor%27s_algorithm

Post Quantum Cryptography II/II

- ▶ Biggest threat right now: „Store now, decrypt later“ attacks for classical key exchanges
- ▶ Classical signatures (RSA, ECDSA, EdDSA, ...) used right now could also be forged but are not threatend by store now, decrypt later. We need to limit their lifetimes!
- ▶ Both attacks can be mitigated by combining classical with PQC algorithms, this is then called „hybrid“. This is not always practical though. More on that when we talk about x509/TLS certificates.

Why is this relevant for Tango?

- ▶ Our approach should use what is now the most secure algorithms which are still practical for our task but also expect to change these once they become broken/obsolete/disallowed
- ▶ This approach is known as „crypto agility“
- ▶ Our **BIG** advantage is that we start on a greenfield⁷

⁷meaning there are no other tango implementations already doing encryption which we have to accommodate

What do we have to protect in a CS?

- ▶ All data on the wire:
 - ▶ Request-Reply⁸ data (attributes, commands, pipes, CORBA attributes/methods) which uses GIOP/IIOP from omniORB as message protocol
 - ▶ Pub-Sub data (events, heartbeat)
- ▶ Identities: A client which connects to a device server a/b/c wants to know that she is really speaking to the real one and not a fake one which was e/f/g 5 minutes ago

⁸tango-rfc speak for CORBA

Cont.

- ▶ Reliability: A bad actor/client on the same network should not be able to make a device server fail. The reverse direction is also true: A bad actor/DS on the same network should not be able to fail a good client.
- ▶ Traceability: A device server can figure out from the logs which client connected to it and what it did

Let's talk about
implementation!

CMake options

- ▶ `TANGO_USE_ENCRYPTION` (defaults to ON, allowed values are ON and OFF): Requires omniORB compiled against OpenSSL and supports encryption for Request-Reply and Pub-Sub.
- ▶ Phasing out unencrypted communication sounds like a good idea, but there will always be the one too-slow machine with an old DS where encryption is not feasible and which has a direct network link
- ▶ Not having to deal with encryption is also nice for the unit/integration tests unrelated to encryption

Request-Reply

- ▶ Transport Layer Security (TLS) is a cryptographic protocol designed to provide secured communications over a computer network⁹.
- ▶ This is what everyone is using on the web (https)
- ▶ omniORB supports TLS natively, so this is what we propose to use for Request-Reply. POC showcased already at the Cyber Security SIG meeting in Lund (March 2024)¹⁰.

⁹https://en.wikipedia.org/wiki/Transport_Layer_Security

¹⁰<https://indico.tango-controls.org/event/237/>

Dependencies

- ▶ Nothing changes with regard to the lowest supported version of omniORB (4.3.0). While 4.3.4 gives us a better interface for configuring TLS, we can just subclass `sslContext` in 4.3.x ($x < 4$) to achieve the same.
- ▶ omniORB is required to be compiled against OpenSSL with `TANGO_USE_ENCRYPTION = ON`.
- ▶ OpenSSL $\geq 3.5.0$ so that PQC support is present without additional configuration hurdles. OpenSSL $\geq 3.5.0$ is available in Debian 13 (trixie) and Ubuntu 26.04 (resolute).

LTS rules

- ▶ As encryption is an **optional** feature, this is not deemed to violate our LTS rules, as Tango can still be used without encryption on older supported tier 1-3 OSes.
- ▶ It's also possible to roll your own omniORB together with OpenSSL on older OSes.
- ▶ And you should be using pixi anyway ;)

TLS

- ▶ Use TLS 1.3 as lowest supported version (as supported by omniORB). The reason is that lower versions should either¹¹ not be used ($< \text{TLS } 1.2$) or only until 2031 (TLS 1.2).
- ▶ 0-RTT is disabled
- ▶ TLS 1.3 only supports ciphers with perfect forward secrecy

¹¹tango-sec-desy.com/literature/BSI-TR-02102-2.pdf

Mutual TLS

- ▶ Mutual TLS (both clients and servers have certificates) is what we suggest:
 - ▶ Only option which verifies that the client has legitimate interest in accessing the device server
 - ▶ Clients without valid certificate clients never reach the application layer (no CORBA requests, no authorization checks), which limits DoS by untrusted clients to the TLS handshake.
 - ▶ The handshake itself (key exchange and server signature happen before the client certificate is checked) and the plain ZeroMQ event ports remain open to DoS.
 - ▶ Having a certificate for the clients also allows for better logging on the DS side (as the client has an identity) and can also be used to bind authorizations to the client requesting it

Pub-Sub: Overview

- ▶ CurveZMQ would be a candidate¹², but it does not support PQC.
- ▶ There is a third-party solution based on ZeroMQ using PQC, but it is not standardized and a pure Go implementation, so this is not an option either.
- ▶ The proposed encryption scheme for the events uses standard AEAD ciphers and exchanges the keys via PQC-secure Request-Reply.
- ▶ Our scheme also has per-entity keys, which CurveZMQ lacks.

¹²different connections from a SUB to a PUB would be possible, the author claimed earlier this is not possible

Pub-Sub: Details I/III

- ▶ Exchange key material via `ZmqEventSubscriptionChange`
- ▶ Keys are per entity: one for the device (interface change events) and one for each attribute (change, periodic, archive, alarm, attribute config, user and data ready events).
- ▶ Keys are rotated by the publisher every N seconds ($N \geq \text{EVENT_RESUBSCRIBE_PERIOD}$). The subscription reply contains the current and the next key, so every subscriber holds a valid key until its next resubscription.

Pub-Sub: Details II/III

- ▶ All subscribers of an entity share the same key. This protects the events against everyone without the key, but there is no origin authentication among key holders: every subscriber can forge events of that entity.
- ▶ Heartbeat events are encrypted with the same scheme using a separate key per device server process (admin device), which is included in every subscription reply of that server. Same rotation rules apply.

Pub-Sub: Details III/III

- ▶ Event encryption follows the Request-Reply connection used for the subscription: If `ZmqEventSubscriptionChange` was called over TLS, the client only accepts encrypted events, there is no fallback to unencrypted events.

End of the
presentation for this
session

Encryption: Support different modes

- ▶ Allow to support different encryption modes: off, mixed and required
- ▶ Device server process gains support for a new CLI flag `-enc=XXX` with the values off, mixed and required. It defaults to mixed when `TANGO_USE_ENCRYPTION = ON` and to off when it is OFF.

cppTango API I/III: Encryption mode

```
1  enum class EncryptionMode : unsigned int
2  {
3      Off = 1,          ///< Never offer or use encrypted transport.
4                        ///< Only supported option when
5                        ///< TANGO_USE_ENCRYPTION is OFF
6      Required = 2,    ///< Require encrypted transport and don't
7                        ///< support unencrypted
8      Mixed = 3        ///< Support encrypted transport but allow using
9                        ///< unencrypted transport explicitly.
10 };
11
12 // Client, needs to be called before the first proxy is created
13 static void ApiUtil::set_encryption_mode(EncryptionMode em);
14 static EncryptionMode ApiUtil::get_encryption_mode();
15
16 // Server, needs to be called before Util::init()
17 static void Util::set_encryption_mode(EncryptionMode em);
18 static EncryptionMode Util::get_encryption_mode();
```

cppTango API II/III: Info Querying

```
1  using StringList = std::vector<std::string>;
2
3  /// Will return a list of key/value pairs with
4  /// encryption-related information
5  StringList ApiUtil::get_encryption_info() const;
6
7  // New DServer command: GetEncryptionInfo
8  DevVarStringArray DServer::get_encryption_info();
9
10 StringList DeviceProxy::get_encryption_info();
11
12 /// Return whether the current Request-Reply connection
13 /// is encrypted.
14 bool Util::is_connection_encrypted() const;
```

cppTango API III/III: Reinit enc. context

```
1  void ApiUtil::reinitialize_encryption();
2
3  // New DServer command: ReinitializeEncryption
4  //
5  /// Reinitializes the TLS context and, among other things,
6  /// re-reads the certificate files from disk
7  ///
8  /// This is the online method for reading in a new certificate
9  /// to avoid or recover from expired certificates.
10 DevVoid reinitialize_encryption();
11
12 void DeviceProxy::reinitialize_encryption();
```

Tango protocol

- ▶ Add a new protocol: `tangos://` (tango secure)
- ▶ In the below table we have the client in the rows and the server in the columns. `x` means that a connection is successful and `-` that it fails.
- ▶ `Off` means a newer `cppTango` version compiled with `TANGO_USE_ENCRYPTION`, a version with encryption support but manually turned off, or older Tango versions.

Explicitly requesting a secure connection

1 TRL: tangos://a/b/c

2

3	Client/Server Required Mixed Off
4	----- :-----: :-----: :---:
5	Required x x -
6	Mixed x x -
7	Off - - -

Opt 1: Requesting an insecure connection

(with automatic promotion)

```
1  TRL: tango://a/b/c
```

```
2
```

```
3  | Client/Server | Required | Mixed | Off |
4  |-----| :-----: | :-----: | :---: |
5  | Required      |         x         |         x         |         -         |
6  | Mixed         |         x         |         x         |         x         |
7  | Off           |         -         |         x         |         x         |
```

Opt 2: Requesting an insecure connection

(without automatic promotion, rejected alternative)

```
1  TRL: tango://a/b/c
```

```
2
```

```
3  | Client/Server | Required | Mixed | Off |
4  |-----| :-----: | :-----: | :---: |
5  | Required      |      -  |      -  |      -  |
6  | Mixed         |      -  |      x  |      x  |
7  | Off           |      -  |      x  |      x  |
```

Tango protocol: Automatic promotion

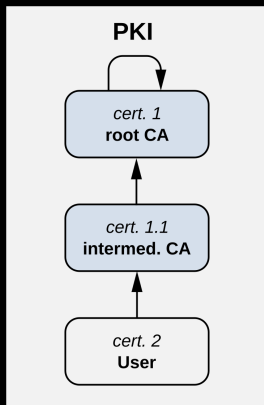
- ▶ The promotion from `tango://` to `tangos://`, if available, allows for more connections in the matrix (7 vs 4) so we propose to add that
- ▶ No protocol prefix, i.e. TRL without protocol (`tango://`), defaults to `tango://` as before, but automatic promotion might change this to `tangos://`
- ▶ If an encrypted connection attempt between client and DS fails, there is no connection at all, this means:
 - ▶ There is **no** downgrade to an unencrypted connection
 - ▶ Same for promoted connections
 - ▶ We basically only use the SSL endpoints from the IOR and ignore the TCP ones

Introduction to x509/TLS certificates

- ▶ When you build up an encrypted communication channel, how do you know that the other side is really who you think it is?
- ▶ TLS solution: cryptographic certificates which are signed

x509/TLS certificates Cont. I

- ▶ Idea: You (without any reason) trust the root CA (1) which is self-signed, and this signs the intermediate CA (1.1) and that signs the leaf certificates (2)
- ▶ This means if you get the certificates, you can verify that the leaf was signed (transitively) by the root CA



x509/TLS certificates Cont. II

- ▶ As we are doing mutual TLS both client and device server process certificates need to be signed by the same root CA
- ▶ We write the device server name(s) into the server certificate and then the client knows that the other side really is who she thinks
- ▶ The server only knows that it is a legitimate client
- ▶ There is a way to mark „lost“ certificates as invalid (CRL lists)

Environment variables I/IV

- ▶ Here we use the term „environment variable“ in the Tango sense, so these can also be present in the usual configuration files.
- ▶ `TANGO_ENC_MODE`: Set the encryption mode, valid values are `required`, `mixed`, `off`, unknown values result in a fatal error. This allows using encryption with zero code changes.
- ▶ `TANGO_ENC_CERT_AUTHORITY_PATH`: Absolute path to a file with one or more CA certificates (PEM format).

Environment variables II/IV

- ▶ `TANGO_ENC_CERT_PATH`: Absolute path to a file with the private key and certificate (PEM format). In case the private key is encrypted, the password must be present in `TANGO_ENC_CERT_PASSWORD`.
- ▶ `TANGO_ENC_CERT_PASSWORD`: Password for the encrypted private key

Environment variables III/IV

- ▶ `TANGOS_HOST`: List of databases, same syntax as `TANGO_HOST`, the only difference being that all the listed ports must be TLS secured. For `mixed/required` clients this needs to be present. If it is not present, we fall back to `TANGO_HOST` but then expect all the TDB ports to be TLS secured¹³.
- ▶ `TANGO_ENC_CRL_PATH`: Absolute path to a file with the certificate revocation lists (PEM format)

¹³Not really cool but needed

Environment variables IV/IV

The environment variables

TANGO_ENC_CERT_AUTHORITY_INLINE,

TANGO_ENC_CERT_INLINE,

TANGO_ENC_CRL_INLINE allow passing the certificates and CRLs (PEM format) without requiring file system access. It is a fatal error when both XXX_PATH and XXX_INLINE are present.

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

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

Environment variables for tweaking

- ▶ All optional, allows for future tweaking
- ▶ TANGO_ENC_ALGO_GROUPS: TLS groups, defaults to
X25519MLKEM768:SecP256r1MLKEM768
:SecP384r1MLKEM1024
- ▶ TANGO_ENC_ALGO_CIPHER: Cipher suites, defaults to the OpenSSL defaults. Will be used for Pub-Sub as well.
- ▶ TANGO_ENC_ALGO_SIG: Signature algorithms including certificate signature algorithms, defaults to the OpenSSL defaults.

End of the
presentation for this
session

x509/TLS certificates

- ▶ Every Tango process only ever has one ORB and thus one set of certificates. We need to distinguish clients, which never run a device server, from device servers, which are both servers and clients. We suggest¹⁴ using short-lived certificates for Tango, e.g. something like 45 days.
- ▶ There will be tooling made available for the whole certificate life cycle
- ▶ Tango is only a user of certificates, nothing in Tango itself handles certificates other than reading, sending and checking them

¹⁴Really only a suggestion, nothing is enforced

Common certificate properties

- ▶ version: Only use v3 certificates¹⁵
- ▶ serialNumber: random and unique, this makes every certificate unique
- ▶ subject DN: CN=<holder>, i.e.
dserver/exec/instance for device servers and the service name for clients. The CN stays the same across renewals and is informational only (logs, tools), all checks are done exclusively on the SAN.

¹⁵Introduced in 2008

CA certificate

We strongly recommend a dedicated CA, or a dedicated intermediate CA, for Tango. This allows keeping the root CA offline and issuing Tango certificates from an intermediate CA only.

Extensions:

- ▶ Basic Constraints (critical): CA=TRUE, pathLenConstraint=0 for the CA issuing Tango certificates
- ▶ Key Usage (critical): keyCertSign, cRLSign
- ▶ Subject Key Identifier: keyIdentifier
- ▶ Authority Key Identifier (intermediate CAs only): keyIdentifier

Client/Device server process certificates

Extensions:

- ▶ Basic Constraints (critical): CA=FALSE
- ▶ Key Usage (critical): digitalSignature
- ▶ Extended Key Usage (critical): - clientAuth (pure clients), clientAuth and serverAuth (device server process)
- ▶ Subject Alternative Name (SAN, pure clients don't have that): -
URI=tango:dserver/exec/instance,
URI=tango:a/b/c, URI=tango:e/f/g ... (list of devices in domain/family/member format without wildcards, aliases must not be used here)
- ▶ Authority Key Identifier: keyIdentifier

x509/TLS certificates Cont. I

Reasons to encode the device name in the certificate:

- ▶ Clients check the SAN of the server certificate once per connection and check for every device accessed over that connection that its name is contained in it.
- ▶ The TDB will do a poor mans authorization approach and check that on all commands which export a device/server or write device/attribute properties that the device/server name mentioned in the command argument is in the SAN. If the TDB uses authorization, this particular check is disabled.

x509/TLS certificates Cont. II

- ▶ An attacker **cannot** reuse a stolen certificate from server A for server B

Certificate Revocation

- ▶ We support certificate revocation lists (CRLs) for clients and servers via `TANGO_ENC_CRL_PATH`. OpenSSL has no default location for CRLs and does not download them, so distributing the CRL file is up to the facility. Therefore the certificates carry no CRL Distribution Points extension. The file, or `TANGO_ENC_CRL_INLINE`, is re-read when calling `ReinitializeEncryption`.
- ▶ The lifetime of the CRLs (`nextUpdate`) needs to be determined by the facility. The CRL file must be updated before that time; an expired or missing CRL lets every TLS handshake fail.

Certificate Revocation Cont.

- ▶ CRLs are only checked during the TLS handshake; already established connections are not affected by an updated CRL.

Certificate Renewal

- ▶ Generate a new key pair on each renewal
- ▶ Replace the certificate files, if using `TANGO_ENC_CERT_PATH`, or update the contents of `TANGO_ENC_CERT_INLINE`
- ▶ Call `ApiUtil::reinitialize_encryption()` for clients and `ReinitializeEncryption()` for device server processes. Alternatively restart the process.

End of the
presentation for this
session

Have a look at attack scenarios

- ▶ 1. Mallory in the same network
- ▶ 2. Mallory has a stolen client certificate
- ▶ 3. Mallory has a stolen server certificate
- ▶ 5. Mallory has full access to the machine running a device server (not AuthDS/TDB)
- ▶ 7. Mallory has full access to the machine running the TDB or its DBMS
- ▶ 8. Mallory can create new valid certificates (access to PKI)
- ▶ 10. Mallory holds the event key of an entity

End of the
presentation for this
session

Finalizing discussion

- 2 TB's SIG meeting plan
- 3 Why encryption for network messages?
- 4 Common digital cryptography concepts
- 5 Cryptography is always on the move I/II
- 6 Cryptography is always on the move II/II
- 7 Post Quantum Cryptography I/II
- 8 Post Quantum Cryptography II/II
- 9 Why is this relevant for Tango?
- 10 What do we have to protect in a CS?
- 11 Cont.
- 13 CMake options
- 14 Request-Reply
- 15 Dependencies
- 16 LTS rules
- 17 TLS

- 18 Mutual TLS
- 19 Pub-Sub: Overview
- 20 Pub-Sub: Details I/III
- 21 Pub-Sub: Details II/III
- 22 Pub-Sub: Details III/III
- 24 Encryption: Support different modes
- 25 cppTango API I/III: Encryption mode
- 26 cppTango API II/III: Info Querying
- 27 cppTango API III/III: Reinit enc. context
- 28 Tango protocol
- 29 Explicitly requesting a secure connection
- 30 Opt 1: Requesting an insecure connection
- 31 Opt 2: Requesting an insecure connection
- 32 Tango protocol: Automatic promotion
- 33 Introduction to x509/TLS certificates

- 34 x509/TLS certificates Cont. I
- 35 x509/TLS certificates Cont. II
- 36 Environment variables I/IV
- 37 Environment variables II/IV
- 38 Environment variables III/IV
- 39 Environment variables IV/IV
- 40 Environment variables for tweaking
- 42 x509/TLS certificates
- 43 Common certificate properties
- 44 CA certificate
- 45 Client/Device server process certificates
- 46 x509/TLS certificates Cont. I
- 47 x509/TLS certificates Cont. II
- 48 Certificate Revocation
- 49 Certificate Revocation Cont.

- 50 Certificate Renewal
- 52 Have a look at attack scenarios