Universal Toolbox for Trust-Enhancing Technologies

About this document

subjects/goals:
– GA4GH 12th Plenary Connect session: Towards GA4GH-powered SPEs/TREs
–– proposing a GIF project around data security:
––– categorical and functional analysis of various trust-enhancing technologies
––– formal language for trust-enhancing tech definition through trust relations
––– universal middleware architecture => toolbox
– formalise: attested TLS protocol and other technologies

NOTE: The document has been shifting from `**PETs**` term (Privacy-Enhancing Technologies) to `**Trust-Enhancing Technologies**` to better fit the goal of describing a universal matrix of protection aspects as it broadens the scope beyond privacy to include encryption's role in ensuring **identity**, **integrity**, and **policy** enforcement, where **data privacy** is one of several possible protection aspects within a larger framework of security.

Contributing: We invite contributors to review the document's goals and framework, focusing on enhancing the **Universal Toolbox for Trust-Enhancing Technologies**. Provide suggestions via comments, ensuring alignment with a focus on identity, integrity, and policy enforcement across hardware, software, and data in all possible contexts. Contributions should adhere to the aim of improving the functional and categorial analysis within this universal framework.

Contributors: Pavel Nikonorov, Ravi Kiran Mahankali, Alex Kanitz

Abstract / Summary

- policy/privacy <=> multi-party environment
- policy <=> trust
- trust definition
- encryption = trust establishment tool
- toolbox of trust-enhancing encryption technologies
- operationalise trust on the infrastructure level

The concept of privacy implies a context where one's information could be observed, accessed, or infringed upon by other parties. The party that shares data with another one mandates its privacy policy that outlines how information should be collected, used, shared, and protected. These policies must be respected and preserved by the receiving party, which requires trust from one to another. Trust characterises the level of confidence in the receiving party's ability to perform certain functions or services correctly, fairly and impartially, along with the assurance that the entity and its identifier are genuine.

Encryption technologies are instrumental in establishing self-sustaining trust relationships between different parties, as well as helping to prove and verify identity and authenticity. This paper presents an architecture concept of a Universal Toolbox for Trust-Enhancing Technologies (TETs) that implements a formalised matrix of security aspects allowing various TETs to be operationalised programmatically as a policy at the infrastructure level.

Notably, the cryptographically-backed protection of some of these aspects is enabled by technologies such as Homomorphic Encryption and Trusted Execution Environments (TEEs). This toolbox can be implemented within a security policy manager, allowing each cluster host to manage its hardware, data and software usage policies within a multi-domain environment, and promises practical, comprehensive, and enforceable policy protection across all data and software states and their interactions in federated environments.

Introduction

Alice <== TET_protocol ==> Bob

TODO: practically,there is always one more party – Eve who manages the network infra. that's why we need data-in-transit confidentiality protection (SSL/TLS)

Both Alice and Bob can possess their digital assets (hereinafter 'secrets') and manage their privacy policy (access control).

The secrets can be categorised into:

- persistent: keys, passwords, data, software sources or executables;
- temporary: tokens, nonce.

There are three states of secrets: at-rest, in-transit, and in-use.

Any TET protocol consists of steps/actions and related handshake models. Possible steps may be categorised into the following possible actions:

- generating secrets (explicitly or based on other secrets);
- sending/receiving secrets with another party (Alice/Bob or software agents);
- verifying secrets;
- storing secrets into controlled storage systems;
- retrieving secrets from the controlled storage systems;
- executing/stopping software agents.

Within a single TET protocol, steps involving secret generation or verification, as well as sending, can leverage other TETs.

Protocol steps that verify received secrets are intended to lead to the trust state update for the receiving party. Trust states may be categorised on:

- Undefined: not yet verified.
- Untrusted: verified with a negative result.
- Self-sustaining trust: relies solely on cryptography (homomorphic, SMC).
- Inferred trust: a logical consequence based on a system of other verifications/pivots, relies on trust anchors such as CA, TEE CPU vendor or software code auditor, which may be identified by a signature/domain;

Trust is non-transitional by its nature and always comes within the context. TETs provide a mechanism of assurance of securing attributes such as identity, integrity, and policy enforcement of subjects (e.g., hardware, software, or data) in various contexts. Depending on the method, the context might imply a local or remote subject location.

security attributes / protection aspects:

- **identity (id)** refers to the origin and authenticity of hardware, software, or data, ensuring that the subject can be uniquely identified and verified.
- **integrity (int)** is also applicable to hardware, software, or data, and means that it remains in its original, unaltered state, free from unauthorised modifications throughout its lifecycle;
- **policy (pp)** applies in various dimensions to:
    - **hardware design:** defines the behaviour and operational rules of physical components, as outlined in their technical specification. This includes how hardware handles data, communicates with other systems, and enforces security measures (e.g., access control, encryption measures, integrity checks, remote attestation).
    - **software licence**: the software developer or intellectual property rights owner's policy on how the software can be distributed, modified, and used, as well as by who, and on which terms.
    - **software design**: defines the effective software's policy on how it collects, stores, analyses, erases, and protects data, including the measures it has to preserve following the data owner's privacy policy.
    - **data privacy**: the data owner's policy on how their data can be collected, stored, analysed, or erased.

In remote contexts, some TETs might help Alice to ensure the security of data, software, or hardware in Bob's infrastructure, even when Alice does not have direct control over it.

security attr.	**subject/object: hardware, software, or data**	**context**
identity	origins, genuineness, authenticity	each protection aspect can be considered in both **local** and **remote** context
integrity	correctness and consistency without unauthorised modification	
policy	hardware or software-designed behaviour; data owner's policy; software licence;	

Note: the protection of data integrity and confidentiality (which is one of the privacy policy aspects) while it's in transit effectively spans into a `remote` context if it is transmitted over a 3rd-party network infrastructure.

Secrets revocability

Trust-Enhancing Technologies

Protection aspects notation:
[aspect:id|int|pol|*]-[subject:hw|sw|da|*]-[location:loc|rem|*] (* – all)
Meaning: `aspect` is being protected for the `subject` in the `location`.

Category	Function	Algorithms	Implementations	Protection Aspects
Hash Functions	Converts data into a fixed-size hash value for data integrity.	SHA-256, MD5, SHA-3	OpenSSL, GnuPG, Libsodium	*-*-loc
Symmetric Encryption	Uses the same key for both encryption and decryption	AES, DES, 3DES	OpenSSL, NaCl, Libsodium	*-*-loc
Asymmetric Encryption	Encryption public key for and a private key for decryption	RSA, ECC, DSA	OpenSSL, GnuPG	id-da-rem, int-da-rem
Quantum Encryption	uses quantum mechanics to securely generate and distribute encryption keys, ensuring tamper-evident communication	QKD	OQS, QuISP, ID Quantique	id-da-rem, int-da-rem
Homomorphic Encryption	Combines the strengths of symmetric and asymmetric encryption.	Paillier, RSA, BFV, BGV, Gentry's Scheme, LTV, NTRUEncrypt, CKKS	MS SEAL, IBM HELib, PALISADE, Microsoft Seal	int-da-rem, pol-da-rem, id-da-rem
Zero-Knowledge Proofs (ZKPs)	allows Alice to prove to Bob that a statement is true without revealing any information beyond the validity of the statement	zk-SNARKs, zk-STARKs	ZoKrates, snarkjs, StarkWare, zkSync	id-da-rem, int-da-rem
Secure Multi-Party Computation	Allows parties to jointly compute a function over their inputs while keeping those inputs private.	Garbled Circuits, GMW	EMP-toolkit, MP-SPDZ, ABY, Sharemind	int-da-rem, pol-da-rem
Distributed Machine Learning	Enables decentralised training of machine learning models across multiple nodes while ensuring data privacy and security	Federated Learning, SWARM Learning,	Flower, PySyft, TensorFlow Federated	int-da-rem, pol-da-rem, id-da-rem
Differential Privacy	Ensures individual data cannot be reverse-engineered or identified within a dataset, even when statistical analysis is performed.	Laplace Mechanism, Gaussian Mechanism, Exponential Mechanism	PySyft, Google DP, IBM DiffPriv, MS SmartNoise, PyTorch Opacus	int-da-rem, pol-da-rem, conf-da-rem
TEEs, TEE Remote Attestation	Provides hardware-isolated and remotely verifiable environments to execute sensitive computations securely.	RATS, DICE, TEEP, TDISP	Intel SGX/TDX, AMD SEV-SNP, ARM TrustZone, Amazon Graviton[1], NVIDIA H100	id–hw-*, id-sw-*, int-*-*
Attested TLS	Allows to communicate with AI in clouds without revealing all input data, as well as the AI software and models itself	IRA-TLS, TLS-a	CCC-PoC, GENXT confido	id-*-*, int-*-*, pol-*-*

Both Alice and Bob can control the execution of one or more software agents or storage systems. Software agents operating within TEE effectively become independent data-controlling parties that, similar to Alice and Bob, can possess their own secrets and manage their privacy policy.

[diagram: Alice/Bob executing TEE-based software agents / Trust Domains]

TEEs stand out from all other technologies by allowing software agents to:

- have remotely verifiable identities;
- possess their own secrets;
- control privacy policy; // thanks to hardware integrity protection and RA
- confidentiality. // thanks to memory isolation

universal trust-establishing protocol: definition/legend

SSS, {SSS^x, SSS^y}, [SSS, KKK] – secret, keypair/keychain, keystore object
AAA – software (sources or executable)
XXX – a party/agent controlling its data privacy policy // trust domain;
trust contexts = TC:

- HI = hardware integrity
- MI = memory isolation // data exposure fully depends on PP
- SI = software integrity // should we split the code itself and the code that is being executed?
- PP = privacy preservation // strict purpose-limitation
- SA = secret authenticity

N. XXX sends[PET] SSS to **YYY** // protocol's N step, sending secret using PET
N. AAA::XXX generates SSS // **AAA** uses **XXX** subsystem being in a single trust domain
N. XXX gets/puts SSS from/to **YYY::VVV** // **VVV** is a **YYY** subsystem
N. XXX runs AAA => **YYY** // executing **YYY software agent**
N. XXX verifies[PET] SSS^x using SSS^y // compute step resulting in (examples):

a. @: **XXX** to SSS^x : TC^1=1 // trust pivot relying on the (@) verification step
b. *: **XXX** to SSS^{x2}: TC^2=1 // explicit (*) trust within the TC context
c. a: **XXX** to SSS^{x3}: TC^3=1 // trust pivot relying on (a) TC^1-trust
d. @: **XXX** to SSS^{x4}: TC^4=1, TC^1=1 // inferred trust
e. **XXX** to **YYY**: (TC)=0 // no trust to **YYY** within the TC context
f.

note: can XXX explicitly trust some SSS? if it's its secret?

Attested-TLS protocol (aTLS)

The aTLS protocol (“a” stands for “attested”) involves multiple trust pivots, allowing Alice to establish a single trust domain within the software agents executed on Bob’s side inside hardware-isolated and verifiable TEEs. The trust context encompasses hardware integrity, memory confidentiality, software integrity, privacy preservation integrity, and the authenticity of all secrets involved. Alice maintains a zero-trust model for Bob and Bob’s software agents.

The use of a TEE can provide Alice with both a verifiable, cryptographically-backed “fingerprint” of Bob’s software agents, as well as a proof of its memory isolation (providing confidentiality and integrity assurances). However, the TEE does not provide insights into how remote software agents deal with Alice’s data before, during and after processing. For this reason, Bob’s software should be well-known to Alice and identified by its hash sums. Hereafter, “HashBox” is the operating system distributive solely designed to execute TES (Task Execution System) and aTLS server agent, while restricting any external access for Bob including SSH and serial console. Similarly, TES is designed to execute tasks without exposing any secrets outside of the TEE boundaries.

The following outlines an aTLS protocol implemented for executing Alice’s tasks within a TEE-VM-based environment on Bob’s side. In this version, Bob’s hypervisor is excluded from the TCB (Trusted Computing Base) for Alice. The only explicit trust anchor in Alice’s TCB is the CPU vendor of the processors deployed in Bob’s infrastructure. This trust includes confidence that the CPU(s) is implemented strictly according to its specification and that the VCEK (Versioned Chip Endorsement Key) infrastructure, hosted by the vendor, remains uncompromised.

1. **Developers** generate InfraSoftware = [OVMF/OpenHCL/vTPM, HashBox, TES]
    a. @: **Developer** to InfraSoftware: SI=1, PP=1
2. **Developer** generates[HASH] InfraSoftwareMeasurements of InfraSoftware
    a. @: **Developer** to InfraSoftwareMeasurements: SA=1
3. **Auditor** verifies InfraSoftwareMeasurements against InfraSoftware
    a. @: **Auditor** to InfraSoftware: SI=1, PP=1
    b. a: **Auditor** to InfraSoftwareMeasurements: SA=1
4. **Auditor** signs & puts InfraSoftwareMeasurements into TRS
5. **Bob** verifies HashBox against **TRS** // trusted repository service, hash-sums store
    a. *: **Bob** to **Auditor**: SA=1 // additional context needed?
    b. a: **Bob** to Auditor.Signature: SA=1
    c. @: **Bob** to HashBox: SI=1, PP=1

6. **Bob** runs HashBox on **CVM** => **HashBox** // HashBox became an **agent** in the Confidential VM
    a. *: **Bob** to **CPU**: HI=1
    b. a: **Bob** to **VCEK-INFRA**: SA=1
    c. a: **Bob** to **HashBox**: HI=1, MI=1, SI=1, PP=1
7. **HashBox** runs OpenHCL/vTPM in **CVM::VMPL0** => **TPM** // OpenHCL is a VM FW
8. **TPM** generates {TPM_EK, TPM_AK} // endorsement and attestation keys
9. **HashBox::CPU** generates TEE_Report={Measurement, TPM_AK}
10. **HashBox::CPU** puts TEE_Report into **TPM::NVRAM**
11. **HashBox** runs TES => **TES**
12. **Alice** generates ClientNonce
13. **Alice** sends[LE-TLS] ClientNonce to **TES** // regular TLS using let's encrypt
    a. **Alice** to **TES**: HI=0, MI=0, SI=0, DC=0, PP=0, SA=N/A // zero trust
14. **TES** generates[TLS] TLSkeypair={TLSpubkey, TLSprivkey}
    a. *: **TES** to TLSkeypair: SA=1
15. **TES** generates SuperNonce using [TLSpubkey, ClientNonce]
16. **TES::TPM** generates TPM_Quote using SuperNonce
17. **TES** gets TEE_Report from **TPM::NVRAM**
18. **TES** generates Evidence = [TEE_Report, TPM_Quote, TLSpubkey]
19. **TES** sends[LE-TLS] aTLS_response=[Evidence, TLSpubkey] to **Alice**
20. **Alice** retrieves TEE_VCEK from **VCEK-INFRA**
    a. *: **Alice** to **VCEK-INFRA**: SA=1 // essential explicit trust
    b. a: **Alice** to VCEK: SA=1
21. **Alice** verifies[SHA/RSA/ECDCA-TBA] Evidence.TEE_Report using VCEK
    a. @: **Alice** to TEE_Report: SA=1
    b. @: **Alice** to **CVM**: HI=1, MI=1, SI=1
22. **Alice** verifies[HASH] Evidence.TEE_Report.Measurement against **TRS**
    a. @: **Alice** to HashBox: PP=1
    b. @: **Alice** to **CVM::HashBox**: HI=1, MI=1, SI=1, PP=1
23. **Alice** generates[SHA] SuperNonce using [Evidence.TLSpubkey, ClientNonce]
24. **Alice** verifies Evidence.TPM_Quote using [Evidence.TEE_Report.AK, SuperNonce]
    a. @: **Alice** to Evidence.TPM_Quote: SA=1
    b. a: **Alice** to Evidence.TPM_Quote.PCRs: SA=1
    c. a: **Alice** to aTLS_response.TLSpubkey: SA=1
25. **Alice** verifies[HASH] Evidence.TPM_Quote.PCRs against **TRS**
    a. @: **Alice** to HashBox: SI=1, PP=1
    b. @: **Alice** to OpenHCL/vTPM: SI=1, PP=1
    c. ab: **Alice** to **HashBox**: HI=1, MI=1, SI=1, PP=1 // boot_aggregate verification
    d. @c: **Alice** to **TES**: HI=1, MI=1, SI=1, PP=1 // IMA-LOG & PCR10
26. **Alice** establish aTLS with **TES** using TLSpubkey
    a. @: **Alice** to aTLS: HI=1, MI=1, SI=1, DC=1, PP=1, SA=1
27. **Alice** sends[aTLS] [**TRS**.TASK, DATA] to **TES**

    a. @: **Alice** to TASK: PP=1
28. **TES::TASK**(DATA) generates RESULT
29. **TES** sends[aTLS] RESULT to **Alice**
    a. @: **Alice** to RESULT: SA=1, MI=1, PP=1

universal toolbox: data structures

```
enum TrustType [explicit, self-sustaining, inferred];

struct TrustPivot {
      type: TrustType,
      a: Agent, // always Agent
      b: Object, // Agent or Secret
      context: Vec<TrustContext> // clarify
}

struct ProtocolStep {
      a: Agent, // always Agent
      b: Object, // Agent or Secret
      c: Optional<Object>, // using c or against c - same
      action: Action, // send, execute, generate, verify, establish
      action_protocol: Optional<PET>, // SHA256, AES, TLS, aTLS
      action_parameter: Secret,
      trust: Vec<TrustPivot>,
}

// example definition
aTLS: PET = {
      name: “aTLS”,
      encryption-type: hybrid, // just meta-information
      trust-context: {
            id-*, int-*, pol-*   // structure to be defined
      },
      protocol: Vec<ProtocolStep>,
            // protocol as a set of steps/actions and handshake models:
            // Each step leads to the trust model update
}
```

universal toolbox: sample entries

```
toolbox = [{
      technology: aTLS,
      vulnerabilities: [], // concept/design-level
      implementations: [{
            vendor: “GENXT LTD”,
            name: “confido”,
            link: {
                  client: “trs://somehost.org/genxt/confido-client”,
                  server: “trs://somehost.org/genxt/confido-server”,
            },
            vulnerabilities: [], // implementation-level
      }, {
            // could be other implementations
      }]
}, {
      "technology": "Homomorphic Encryption",
      "vulnerabilities": [],
      "implementations": [{
            "vendor": "Microsoft",
            "name": "Microsoft SEAL",
            "link": {
                  "client": "trs://somehost.org/microsoft/seal-client",
                  "server": "trs://somehost.org/microsoft/seal-server"
            },
            "vulnerabilities": []
      }, {
            "vendor": "IBM",
            "name": "HELib",
            "link": {
                  "client": "trs://somehost.org/ibm/helib-client",
                  "server": "trs://somehost.org/ibm/helib-server"
            },
            "vulnerabilities": []
      }]
}]
```