# A Maturity Model for Data Trusts

Richard Stewing[1][0000−0001−5840−2131] and Falk Howar[1,2][0000−0002−9524−4459]

[1] TU Dortmund University, Otto-Hahn-StraSSe 14, 44227 Dortmund, Germany
{richard.stewing,falk.howar}@tu-dortmund.de
[2] Fraunhofer ISST, Emil-Figge-Str. 91, 44227 Dortmund, Germany

**Keywords:** Data Trust, Verification

**Abstract.** We design a three-pillar five-level maturity model for data trust with respect to the European regulation on data management. Pillar one focuses local data management. Local data management regards the correctness of logs, usage policies of data, and local access to data. The second pillar of interest is how data is shared. Specifically, it handles the assessment for the data recipient. Lastly, we define an area for GDPR-compliance. GDPR mostly defines right of data subjects with respect to their personal data. These induce processes for entities handling personal data with regards to providing the personal information to a subject and how to rectify or delete data collected. The five levels locate the responsibility to ensure the regulation with the individual, the organization, technical infrastructure, and formal methods. We also provide resources from the literature for Level 3 (Technical Methods) and Level 4 (Formal Methods).

## 1 Project Overview

The European Union has attempted multiple times in the past to structure data economies. Two of the latest attempts are the Data Governance Act (DGA) [20] and General Data Protection Regulation (GDPR) [19] regulating services handling foreign data[3]. The former handling the sharing of (non-personal) data, while the latter discusses the rules for handling personal data and what rights data subjects hold. Based on these regulations, we define a maturity model with three key-areas of concern - local data management, data sharing, and GDPR-compliance.

Local data management focuses on the core components the regulation requires. We focus on three components here:

- Logging of any operation: Any operation like reading, writing, archiving or processing have to specifically logged permanently.
- Usage Policies: Foreign data has policies attached to it to describe what operations can be performed on them. Checking usage policy conformance assures that no unintended processing is executed.
- Local Access to Data: The system also have to note who or what systems have potential access to the data in question.

The maturity of these components can range from *ad hoc* processes to fully automatic and verified implementations.

Data sharing with additional parties is only permitted under specific conditions. On the one hand, the data subject has to have given permission to share its data. Additionally, the recipient has to follow DGA and GDPR themselves and be located in the European Union or an acceptable third country.

Lastly, GDPR-compliance sets specific requirements on the handling of personal data. GDPR protects a data subject's rights regarding their personal data and what they can enforce even after someone else gets access to them. Specifically, they have the right to view what data has been collected, rectify information, or request the deletion of information. All these operations require the implementation of processes to handle the interaction between the data handling service and data subjects.

*Outline.* Section 2 introduces what this documents considers a data trust and describes its structure with terms used in the Data Governance Act and the General Data Protection Regulation. Based on the data trust model from Section 2, Section 3 develops the maturity model. It also provides concrete questions with which to assess a systems current maturity and how to further mature the system with additional techniques and pointers to the literature. To illustrate

[3] We define *foreign data* as data either regarding a different entity or collected by a different entity.

how formal methods can ensure specific properties induced by the regulation, Section 4 shows three examples. Firstly, we show how linear types can prove that a specific method was called on every execution path. Secondly, we use `TLA`$^+$ to specify a simple communication protocol and show that its communication is well-formed, ie., that no unexpected messages arrive at either participant. Lastly, we explain how digital signatures can ensure the origin of a message. Section 5 summarizes the work.

## 2 What is a Data Trust?

Data trusts lack an standard definition in the literature. Every definition scopes the responsibilities of the data trust differently. Furthermore, regulation introduces the term "data intermediation service" with an additional definition that imposes regulation to data trusts [20]. Here are a selection of definitions from the literature:

- "In a legal setting, trusts are entities in which some people (trustees) look after an asset on behalf of other people (beneficiaries) who own it. In a data trust, trustees would look after the data or data rights of groups of individuals" [2]
- "Data trust is a fairly new concept that aims to facilitate data sharing by forcing data users to be transparent about the process of sharing and reusing data. Data trust entails legal, ethical, governance and organizational structure as well as technical requirements for enabling data sharing" [40]
- "A data trust must perform various tasks: It must be able to assign access rights to data, it may or may not need to hold data itself, it must be able to audit whether organizations adhere to their agreed conditions and it must have access to credible tools of enforcement" [9]

All these definitions have "data sharing" in common and define and ensure an access policy. The underlying data is neither about nor owned by the data trust but a beneficiary of the trust. We now define key terms using the same vocabulary and *spirit* as the Data Governance Act and accompanying regulation.

**Definition 1 (Data).** Data *is a fact, an observation, or information in a digital format.*

**Definition 2 (Data Subject).** *A* data subject *or the* subject of data *is the person, object, event, or data that is described by a piece of data.*

**Definition 3 (Data Holder).** *A* data holder *is the natural or legal person controlling and managing the data.*

**Definition 4 (Data User).** *The* data user *is the natural or legal person that has lawful access to the data.*

**Definition 5 (Data Manager).** *A* data manager *is a natural or legal person that upon request of the data holder manages the access from data users to the data holder's data.*

**Definition 6 (Data Manager Client).** *A* data manager client *is a natural or legal person that upon request of the data user requests the access from data trust to the data holder's data.*

We present the relationships between these terms visually in Figure 1. $A$ is a data holder that holds data $a$. Manager $M_A$ manages access to $a$ for $A$. Data user $B$ attempts to access $a$ with a data manager client $M_B$.

As stated above, the literature agrees that data trusts are vehicles for sharing and ensuring access policies. Depending on who we require to ensure the access policies are met, the meaning of the term "data trust" in Figure 1 changes. There are three possible groups of entities that can be settled with that responsibility: the data manager, the data user and its client, and the data manager and client network. We now discuss consequences for these situations.

*The data manager operates as a data trust.* In the first case, the data manager takes on the additional responsibility of the data trust, represented through the solid box in Figure 1. That is, it shares data and ensures that usage policies are met. The usage has to be provided by the data manager client $M_B$ and the data manager has to trust the provided information. In this scenario, the data trust bases its decisions completely on the information provided from the outside. A certification for a data trust considers these decision procedures for this definition of a data trust. This is the first intuitive thought when considering a data trust, an entity managing and ensuring usage of a costumers data.

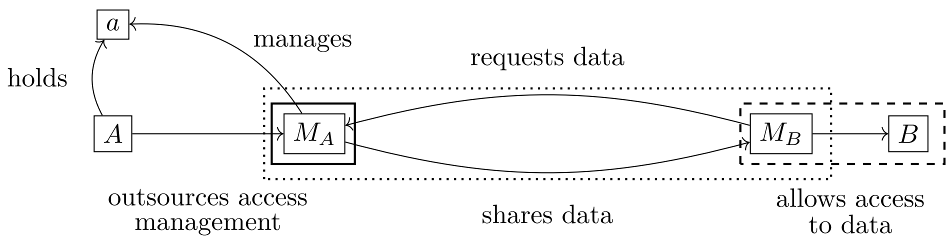


**Fig. 1.** $A$ is the data holder holding data $a$ that is under management at $M_A$. Data user $B$ is requesting $a$ via the users infrastructure for communicating with data manager $M_B$.

*The data user and its client operate as a data trust.* The second case moves the requirements to the other side of the interaction. The dashed box in Figure 1 shows this scenario. For the certification as a data trust, the data user would get audited to ensure that the usage reported by client $M_B$ accurately represents the actual usage of $B$. A drawback of this definition is that certification in this scenario requires auditing of non-public intellectual property in data user $B$ and required domain knowledge. A restriction to only the client $M_B$ is also possible but would only certify that information passed through the client is not altered and executes sharing operations correctly. The ISST[4] follows this definition.

*Data manager and client cooperatively operate as a data trust.* The last case constitutes of parts of both sides of the interaction. In this case the data manager and its client create a certified network for trusted data exchange. As in the first case, the manager is validated to take the right decisions and execute all required steps. Additionally, the client is certified as in the second scenario. Lastly, the communication is also part of the data trust and protocols are inspected for correct behavior. On the other hand, the data user $B$ is not audited for the data trust specification. That is, it is feasible for a malicious data user to report wrong information to its client that leads to misbehavior.

None of the three definitions for data trust provides the full spectrum of guarantees. In the first case, we only know that $M_A$ decides to share data correctly when provided correct information. The second scenario provide guarantees that the usage is correct but would require deep knowledge of the operation of data users. For the last case, we can guarantee that the interaction between manager and client is correct but still need correct information from the data user.

From our perspective, the core of the data trust is best encapsulated in the third scenario. It requires additional certification for the data user to derive actual usage policy information. But this certification does not describe the core functionality of the data trust, ie., sharing data under usage agreements. Therefore, we define data trusts as a cooperative network between data managers and their clients.

**Definition 7 (Data Trust).** *A* data trust *is the cooperative network between and including data managers and their clients.*

The maturity model developed in this work is based on this definition. Adaptation to all definitions is possible as the requirements are not tied to any specific structure.

## 3 Maturity Model

We now present a maturity model for the conformance of a data trust with the DGA [20] and GDPR [19]. We use the definitions stated above. A maturity model consists of three different components. Firstly, we describe the different capabilities necessary for a data trust. Secondly, we define the maturity curve, ie., the different levels of maturity different capabilities can exhibit. Lastly, we provide a way to assess a given system. To that end, we formulated questions for each level and capability.

We define seven capabilities over three core pillars. These are organized to be the three core responsibilities of data trust, ie., local data management, data sharing, and how to handle personal data. Local data management consists of three

[4] https://www.isst.fraunhofer.de

main areas as discussed above. Data sharing requires the data trust to manage sharing permissions, communication with other parties, and to ensure recipient compliance with DGA [20] and GDPR [19]. Handling personal data imposes the provision of an interface for data subjects to view, correct, and delete their data, ie., a data trust has to handle communication with data subjects. These capabilities are categorized and described in Table 1.

| Pillar | Capability | Description |
|---|---|---|
| Local Data Management | Logging | According to the DGA [20] and GDPR [19] operations on and with data have to persistently logged. Logging should include the time, the place, the operation, and a reference to the data the operation used. |
| | Usage Policies | Usage Policies describe how data may be used. An operation executes on data, the system must ensure that the specific operation is allowed to execute on it. |
| | Local Access | One specific usage of data is read access. Before any data is read from the storage medium, it should be checked that the data may be read in the current circumstance. |
| Data Sharing | Permission Management | Sharing data is dependent on whether the data subject has given permission for the data to be shared. This permission has to be validated before sharing can be allowed. |
| | Communication Safety | Communication safety ensures that the communication is secure, eg., encrypted and authenticated, and that all participants only enter well defined states. |
| | Recipient Compliance | DGA [20] and GDPR [19] require that all recipients of shared data also follow the same regulation. Sharing requires that the sharer validates the recipients compliance with the regulation. |
| Personal Data | Data Subject Communication | The regulation allows data subjects to view, correct or delete their data. This requires direct communication with the data holder and processes to react to requests. |

**Table 1.** Data Trust capabilities to meet regulatory requirements of the GDPR [19] and DGA [20].

Our maturity model has five levels. At Level 0 no guidelines and processes are defined. We leave it out of any further discussion because at that level, the regulatory requirements are clearly not met. Level 1 describes minimal requirements to start considering a system as data trust. It puts the requirements on the individual implementer of the system. Level 2 adds organizational oversight to the approach with review systems and intra-organizational accountability. We add technical assurance at Level 3. Here defined technical primitives ensure that requirements are met. Lastly, Level 4 adds formal assurance in the form of formal methods to Level 3's technical assurance. Formal methods are capable to *prove* that the specific requirements are fulfilled. Table 2 shows a general description of the different levels of the maturity curve. It also includes general steps to achieve the next level in the model. We provide more details on each specific capability in Table 3.

Assessing a system's maturity requires to ask whether specific capabilities are correctly implemented. Table 4 provides questions for each capability and level to quickly and correctly assess the system. The questions are incomplete and may require further investigation to correctly judge the maturity.

| Pillar | Capability | Level 1 | Level 2 | Level 3 | Level 4 |
|---|---|---|---|---|---|
| Local Data Management | Logging | Do coding guidelines describe how logging has to be integrated into the implementation? | Do reviews specifically require to check logging operations and their compliance to coding guidelines? | Does the platform implement logging in primitive operations? Does the platform restrict operations such that only logged primitives can be used? Does the platform monitor the execution of all components? | Can we provide all operation traces? Does each include the correct logging operations? |

| Pillar | Capability | Level 1 | Level 2 | Level 3 | Level 4 |
|---|---|---|---|---|---|
| | Usage Policies | Do coding guidelines describe how policy checks have to be integrated into the implementation? | Do reviews specifically require to check policy checks and their compliance to coding guidelines? | Does the platform implement policy checks in primitive operations? Does the platform restrict operations such that only checked primitives can be used? Does the platform monitor the execution of all components? | Can we provide all operation traces? Does each include the correct policy checks each for each operation? |
| | Local Access | Do coding guidelines describe how access checks have to be integrated into the implementation? | Do reviews specifically require to check access rights for operations and their compliance to coding guidelines? | Does the platform implement access checks in primitive operations? Does the platform restrict operations such that only checked primitives can be used? Does the platform monitor the execution of all components? | Can we provide all operation traces? Does each include the correct access rights checks for each operation? |
| Data Sharing | Permission Management | Do coding guidelines describe how sharing permission checks have to be integrated into the implementation? | Do reviews specifically require to check sharing permissions for operations and their compliance to coding guidelines? | Does the platform implement sharing permission checks in primitive operations? Does the platform restrict operations such that only checked primitives can be used? Does the platform monitor the execution of all components? | Can we provide all operation traces? Does each include the correct sharing permission checks for each operation? |
| | Communication Safety | Is the communication protocol only defined in the implementation? Is the implementation required to develop a tool that can communicate with it? | Is there a informal document describing the communication between participant? | Is there a formal model for the communication? Is the document detailed enough to create a corresponding implementation? | Is there a mechanical method to check that the formal model and the implementation behave in the same way? |
| | Recipient Compliance | Not Applicable | Does the recipient ensure that they behave according to regulation? | Has the infrastructure been reviewed by external and reputable sources? | Has the external reviewer been certified by government body? Can it issue official certification? |

| Pillar | Capability | Level 1 | Level 2 | Level 3 | Level 4 |
|---|---|---|---|---|---|
| Personal Data | Data Subject Communication | Is communication organized by a single contact form with free text entry? | Is there a document describing how to communicate with data subjects? | Does the platform organize communication with tools and different communication channels for different issues? | Does the organization review past communication to improve the communication process? |

Table 4: Questions to assess the current maturity level for an implementation.

Level 4 requires the use of formal methods to generate proofs for regulatory requirements or increase the design of the platform. The latter is especially important with regards to the data subject communication. Table 5 collects literature as starting points to integrate formal methods in the design and implementation of a data trust platform. It also includes introductory literature for creating tests as first tool to increase assurance informally. The literature for Levels 1 and 2 mostly considers development processes and how to manage them [6, 21, 43]. Additionally, resources on coding guidelines and code review can provide a starting point to create the necessary resources for a specific organization [3, 22].

In Section 4, we show three examples to explain how formal methods can ensure properties of systems. Firstly, we are going to show how linear types can ensure that a function, eg., the log function, is called before an operation is executed. Secondly, we show the specification of a simple communicating system in `TLA`$^{+}$ and how it ensures that no unsafe state is reached. Lastly, we describe how a digital signature can ensure that a recipient is externally reviewed by a trusted entity.

## 4 Examples

This section shows three examples of how formal methods can create certainty for specific aspects of an implementation. Linear types have the ability ensure dependencies between calculations by restricting the use of values in programs. `TLA`$^{+}$ specify communication protocols formally. This enables automated tools explore the state space and find undesired behavior. Lastly, cryptography enables any recipient to verify the origin of a message.

### 4.1 Linear Types

Linear types as described by Wadler [45] impose restrictions on the use values in a programming language. Each value must be used exactly once. This has to the ability to firstly improve static memory management and, more importantly here, can create a dependency management system in the language.

For example, the function operation(_prf:LogPrf) requires a LogPrf value. And because the only way to create such a value is to call the log function, we are ensured that the log function is called. The linear types ensure that the LogPrf value is used only once. Hence, if we try to call operation again with the same LogPrf value (and having logged only one operation), the compiler will throw an error. Additionally, if we create a LogPrf value and never consume it with an corresponding operation, ie., log an operation that is never executed, the compiler will also throw an error. Therefore, linear types ensure and *prove* during compile time that every operation has a corresponding call to the log function.

An example with the complete code are demonstrated in Figure 4.1. The description above is true for linear types, but `Rust`[5], the example's implementation language, does not implement linear types. `Rust` implements *affine* types. Affine types guarantee that every value is used at most once. That is, affine types ensure that any operation has a log operation, because otherwise a LogPrf value would have been used twice, but they cannot ensure that any log operation also has a corresponding operation executed to it. Secondly, we require that the only way to construct a LogPrf value is through the log function. This requires to hide the constructor with the package infrastructure but this is a standard operation in all mainstream programming languages.

[5] https://www.rust-lang.org

| Level | Description | Next Steps |
|---|---|---|
| Level 0: Unstructured Processes | Level 0 does not provide any guidelines or processes to implement the regulation. A system with maturity Level 0 does not meet regulatory requirements. | The definition of informal guidelines is the first step mature such a system. |
| Level 1: Minimal requirements | Level 1 describes the minimal requirements for each capability. The regulatory requirements are meet by self-organized discipline on an individual level. Notice that for the "Recipient Compliance" capability, this level is not applicable. Instead the other organization has to *as an organization* assess their own capabilities at Level 2. | The next step for the organization is to introduce an inner organizational - multiple eyes principle. This involves creating documentation and review of key components of the system. |
| Level 2: Organizational Assurance | Level 2 introduces internal validation to the products and processes the organization develops. Code reviews require that at least to developers validate a piece of code. Documentation can be read and validated by others internal to an organization. Self assessment to a second party introduces internal pressure to correctly state the organizations capabilities. (Sure :D) | Introducing technical means to ensure specific aspects is the next step. The platform can introduce primitives that ensure that specific aspect are used. Thereby limiting the validation requirements to only those primitives. Formal methods my be checked mechanically for syntactic correctness and provide a uniform platform for communication between engineers. External reviews codify the checks a platform has to undergo to participate in a data sharing network. |
| Level 3: Technical Assurance | Level 3 adds technical assurance to the product and processes. Adding primitives that fulfill regulatory requirements ensures that these requirements are met. Implementing the communication processes according to a formalized specification removes a level of uncertainty. Additionally, using the platform to guide the communication with data subjects ensures that all necessary components are present. | Adding formal methods for verification to the technical assurance removes another source of human error by verifying the implementation. Formal methods can create program traces to ensure the calling of necessary operators. Additionally, formal methods can analyze communication protocols. |
| Level 4: Formal Assurance | Level 4's formal method can verify the technical assurance added in Level 3. Analyzing specification and implementation of components can ensure that implementations meet the specification. Furthermore, specifications can be analyzed to ensure that it behaves as intended and does not exhibit unintended behavior. | Additional formal methods can always be applied to more completely describe the behavior of the system. |

**Table 2.** Data Trust Maturity Curve describes the four levels of maturity. It also includes the conceptual steps to increase the maturity. The concrete state for each capability in each maturity level may be found in Table 3.

### 4.2 Communication Specification

Communication with other systems work along a agreed-upon protocol between all parties. These might not be formally defined and only defined through its implementation but a protocol is *always* present when communicating. Multiple formalism have been defined to describe the communication between systems. The Calculus of Communicating Systems (CCS) [33] and Communicating Sequential Processes (CSP) [25] where first approaches to describe these systems with a fixed number of components. These were later extend to allow the creation of new communication links between components with the $\pi$-calculus [41]. Another approach defines the communication between two (or more) parties as a single system and models it appropriately. This has the advantage that we are able to use an of the shelf specification framework like `TLA`$^+$.

`TLA`$^+$ is an industry strength tool to describe and check systems of varying complexity. It describes models in the *Temporal Logic of Actions*, ie., specifications are logical formulae. An action is a formula that describes the current state of variable with formulae like $x = exp$ and the next state of a variable with $x' = exp$. (Notice the *tick* after the $x$.)

In our concrete example, the full specification is in Appendix A, we define the communication between a data trust and a client. The client sends a request, the data trust responds and the client sends an acknowledgment. (For this example, we abstract away from the concrete value of the request data and use a fixed placeholder.) Both, the client

| Pillar | Capability | Level 1 | Level 2 | Level 3 | Level 4 |
|---|---|---|---|---|---|
| Local Data Management | Logging | Coding guidelines require logging of operations. | Code reviews specifically review logging operations. | The platform infrastructure globally implements logging primitives within core functionality. | Formal methods ensure the execution of logging operations. |
| | Usage Policies | Coding guidelines require policy checks before data usage. | Code reviews specifically review policy checks. | The platform infrastructure globally implements policy checks. | Formal methods ensure policy checks before data access. |
| | Local Access | Coding guidelines require access rights. | Code reviews specifically review access rights. | The platform globally implements access rights. | Formal methods ensure access rights. |
| Data Sharing | Permission Management | Coding guidelines require to check for sharing permission. | Code reviews specifically review sharing permission checks. | The platform globally implements sharing permission checks. | Formal methods ensure sharing rights. |
| | Communication Safety | Communication is defined by the implementation. | Communication is informally described. | Communication is formalized and informally reviewed. | Communication is formally specified and formal methods ensure communication safety. The implementation is checked to correspond to the specification. |
| | Recipient Compliance | Not Applicable | Recipient self assesses compliance. | External review assesses compliance. | External review assesses compliance and issues an official certificate. |
| Personal Data | Data Subject Communication | Communication with data subjects is ad hoc and not defined. | Communication with data subjects is defined but carried out manually. | Communication with data subjects is defined and supported by platform infrastructure. | Communication with data subjects is defined, supported, reviewed, and improved continuously. |

**Table 3.** Maturity levels for Data Trust Capabilities.

and the data trust, have an inbox that can hold exactly one message (1 or 2) or is empty (0). Additionally, the data trust and client are in different states depending on which next operation they are going to take or expecting from the other. We will now take a closer look at the action ReadRequestFromDT.

$$\begin{aligned} \text{ReadRequestFromDT} \triangleq\ & \text{DTState} = 0 \\ & \land \text{ReadFromDT}(1) \\ & \land \text{DTState}' = 1 \\ & \land \text{UNCHANGED}\langle \text{inboxClient}, \text{clientState} \end{aligned}$$

The action is only enabled, ie., can be executed, if the data trust is in the 0 state and we can read a 1 from the data trust inbox. In that case, the data trust transitions to the 1 state and the client's inbox and its state remain untouched. Intuitively, we model that the data trust reads from its inbox and reacts with a state transition. By combining multiple of these actions, we can describe sequences of transitions and in combinations whole behaviors of systems.

Given such an behavior, the $\texttt{TLA}^+$ tool box is able to search the whole state space and prove that specific behaviors never exists. For example, in the present specification, we can show that the data trust inbox never contains an unknown message, ie., the data trust is never blocked. Similarly, we can show that all responses from the data trust to the client's inbox do not block the client. More complex properties can also be encoded.

Formalizing the communication with tools like $\texttt{TLA}^+$ provides a clear definition which can be referenced by implementers to check that their systems behave correctly (or at least up to specification). It also guarantees that the system behaves in known ways as long as other participants also behave in accordance with the protocol.

<table>
<tr><th>Pillar</th><th>Capability</th><th>Technical Methods</th><th>Formal Methods</th></tr>
<tr><td rowspan="3">Local Data Management</td><td>Logging</td><td rowspan="4">– Logging/Monitoring Runtime Systems, eg., ERTS [18]<br>– Program Testing [5, 29]</td><td rowspan="4">– Information Flow Analysis [4, 8, 35]<br>– Linear Types [45]<br>– Functional Verification [7, 14, 24, 26, 28, 32, 37, 44, 46]</td></tr>
<tr><td>Usage Policies</td></tr>
<tr><td>Local Access</td></tr>
<tr><td rowspan="3">Data Sharing</td><td>Permission Management</td></tr>
<tr><td>Communication Safety</td><td>– Automata Specifications [27, 31, 47]<br>– Algebraic Specifications: CSP [25], CCS [33, 34], $\pi$-calculus [41]<br>– System Testing [10, 12, 16]</td><td>– Model Checking [13, 17, 27, 31, 36, 39, 42, 47]</td></tr>
<tr><td>Recipient Compliance</td><td colspan="2">Cryptography (Digital Signature) [1]</td></tr>
<tr><td>Personal Data</td><td>Data Subject Communication</td><td>– User Interface Design [15, 30, 38]</td><td>– Log Analysis [23]<br>– User Feedback [11]</td></tr>
</table>

**Table 5.** Technical and formal methods applicable for Levels 3 and 4.

### 4.3 Ensuring Originality with Digital Signatures

Digital signatures provide the means to provide a proof of origin and are the dual to encryption. This subsection explains the general concept and how it guarantees the origin of a document [1].

We consider the following scenario. Alice has Bob's public key $A_{pub}$ and their own keys $A_{priv}$ and $A_{pub}$. Bob has Alice's public key $A_{pub}$ and their own keys $B_{priv}$ and $B_{pub}$. Figure 3 shows what keys what participants have access to.

For each public/private pair of keys $K_{pub}$ and $K_{priv}$, both keys reverse each other. That is, for every message $m$ we have $K_{pub}(K_{priv}(m)) = m$ and $K_{priv}(K_{pub}(m)) = m$. Also, it is very hard to find the private key (public key) given the public key (private key), ie., given $K_{pub}$ ($K_{priv}$) it is computationally hard to find $K_{priv}$ ($K_{pub}$) such that the above property holds.

Before we discuss how Alice and Bob can exchange signed messages, we will describe how they send encrypted messages. We assume Alice wants to send Bob a private message $m$ such that only Bob can read it. Alice can use *Bob's* public key $B_{pub}$ to encrypt the message to $B_{pub}(m)$. The only way to recover $m$ from $B_{pub}(m)$ is to apply Bob's private key to it, ie., $B_{priv}(B_{pub}(m)) = m$. Since we know that only Bob has his private key (and that it is not easy to calculate from $B_{pub}$), we know that only Bob can execute that operation. Hence, the message is remains private and only Alice and Bob know $m$'s contents.

The case for signing is slightly different. In the case of encryption, our goal is that the recovery of the message is only executable by the recipient. For signing, we want the signing only be possible by the sender. Now assume, Alice wants to send Bob a message $m$ such that Bob knows it was her that sent it. The only operation only Alice can execute is $A_{priv}(m)$. If she sends $A_{priv}(m)$, he can apply *her* public key to it and recover the message.[6] At this point, Bob has the message $m$ and presumably knows that Alice send it.

How can an attacker attempt to send a message $m$ as Alice? He would need to find a message $m'$ such that $A_{pub}(m') = m$. This would essentially[7] attempts to calculate the Alice's private key which is, by definition, computationally hard.

The other possibility is that the attacker sends random messages, ie., he just sends any $m'$ and waits to see how Bob reacts after encryption. But as long as we can distinguish random messages from well-formed messages, we can simply ignore them. Notice that well-formedness requires a certain degree of domain-knowledge.

[6] Everybody who has her public key can recover the message! Signing does not ensure that only the recipient can read the message.

[7] This is technically a simpler problem then finding $A_{priv}$, because finding $A_{priv}$ solves the above problem for *every* message $m$, while the above searches for a solution for a specific $m$. On the other hand, this exactly allow to send a $m$ as Alice. If you want to provide $m$ as an input to that algorithm, you are again calculating $A_{priv}$.

```
// The constructor of LogPrf is hidden in a module.
// The only way to create one, is to call the log function.
enum LogPrf {
    Prf
}

fn log_operation(_msg : String) -> LogPrf {
    println!("Log␣Operation");
    LogPrf::Prf
}

// The operation consumes the LogPrf.
// Therefore, every operation needs its own LogPrf and the creation creates the log.
// It is therefore impossible to execute the operation without creating a log.
fn operation(_prf : LogPrf) {
    println!("Execute␣Operation");
}


fn main() {
    // Create LogPrf which creates the Log.
    let prf = log_operation("First␣log␣entry".to_string());
    //Execute Operation
    operation(prf);
    //A second operation fails without its own log entry.
    //operation(prf);
}
```

**Fig. 2.** An example of using linear types to ensure a logging function is called.

By using this form of digital signature, we can ensure that we know who has send a specific message. We can apply this technique to the regulatory certification. If a external reviewer certifies that a specific service acts according to the regulation laid out in the DGA [20] and GDPR [19], it can sign a machine readable document of the fact and hand it to the service. Upon the request, the service can provide the document and we can check with the reviewers public key that the document is genuine. This approach also has the advantage that the reviewer does not need to provide infrastructure to certify a service for every data exchange. It can simply publish its public key.

## 5 Conclusion

We present a maturity models for the different aspects of a data trust when they implement the DGA [20] and GDPR [19]. We focus on three pillars - data management, data sharing, and personal data. Every pillar has capabilities that must the implemented to some degree.

A proper local data management is prerequisite for any data trust. A data trust that also shares data needs to manage additional permissions, communication with other parties, and ensure the recipient's compliance with the regulation. Handling personal data implies that the data trust has processes to communicate with data subjects and provides them with an interface to execute their rights as data subjects.

The maturity of a service's capabilities is judged across four levels of maturity relevant for data trusts. The initial level pushes the requirements to the individual implementer or operator. The second level moves these to organizational measures, while the third also implements technical restrictions. The fourth level introduces formal methods to validate the technical restrictions of Level 3.

A service that has implemented every capability on the fourth level can provide proofs for most properties required by the DGA [20] and GDPR [19]. The problem of how to validate the behavior of a data trust as a user remains, ie., how can a user validate that data trust that said it delete data, has in fact delete the data? A similar problem exists between all participants in a network and remains an open problem in computer science.

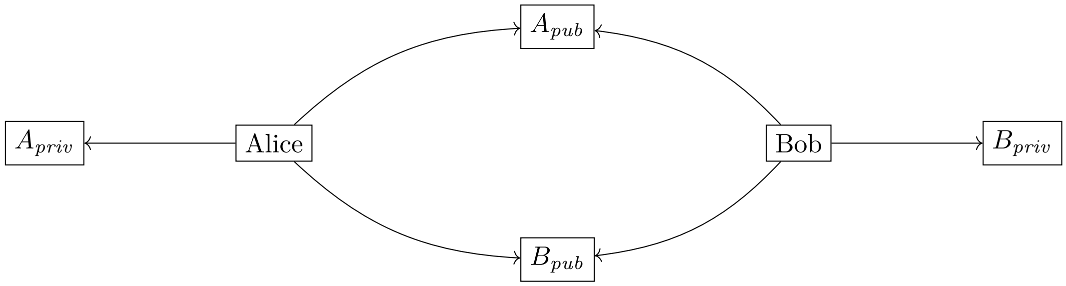


**Fig. 3.** The graphic shows what keys both participants have access to.

## References


1. Anderson, R.: Cryptography, chap. 5, pp. 145–205. John Wiley & Sons, Ltd (2020). https://doi.org/https://doi.org/10.1002/9781119644682.ch5, https://onlinelibrary.wiley.com/doi/abs/10.1002/9781119644682.ch5
2. Anouk, R.: How data trusts can protect privacy. we can't expect people to navigate the confusing world of data collection on their own (2021)
3. Badampudi, D., Unterkalmsteiner, M., Britto, R.: Modern code reviews-survey of literature and practice. ACM Trans. Softw. Eng. Methodol. **32**(4) (may 2023). https://doi.org/10.1145/3585004, https://doi.org/10.1145/3585004
4. Baldoni, R., Coppa, E., D'Elia, D.C., Demetrescu, C., Finocchi, I.: A survey of symbolic execution techniques. ACM Comput. Surv. **51**(3) (2018)
5. Beck, K.: Test driven development. The Addison-Wesley signature series, Addison-Wesley Educational, Boston, MA (Nov 2002)
6. Beck, K., Beedle, M., Van Bennekum, A., Cockburn, A., Cunningham, W., Fowler, M., Grenning, J., Highsmith, J., Hunt, A., Jeffries, R., et al.: Manifesto for agile software development (2001)
7. Becker, M., Regnath, E., Chakraborty, S.: Development and verification of a flight stack for a high-altitude glider in ada/spark 2014. In: Computer Safety, Reliability, and Security: 36th International Conference, SAFECOMP 2017, Trento, Italy, September 13-15, 2017, Proceedings 36. pp. 105–116. Springer (2017)
8. Biere, A., Cimatti, A., Clarke, E., Zhu, Y.: Symbolic Model Checking without BDDs, pp. 193–207. Springer Berlin Heidelberg (1999). https://doi.org/10.1007/3-540-49059-0_14
9. Blankertz, A., Specht, L.: Designing data trusts. why we need to test consumer data trusts now. designing data trusts (2020)
10. Boettiger, C.: An introduction to Docker for reproducible research. ACM SIGOPS Operating Systems Review **49**(1), 71–79 (2015)
11. Bregman, P.: How to ask for feedback that will actually help you. Harvard Business Review (2014)
12. van der Burg, S., Dolstra, E.: Automating system tests using declarative virtual machines. In: 2010 IEEE 21st International Symposium on Software Reliability Engineering. pp. 181–190 (2010). https://doi.org/10.1109/ISSRE.2010.34
13. Chaki, S., Rajamani, S.K., Rehof, J.: Types as models: model checking message-passing programs. In: Proceedings of the 29th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages. pp. 45–57 (2002)
14. Chlipala, A.: Certified programming with dependent types: a pragmatic introduction to the Coq proof assistant. MIT Press (2022)
15. Cohn, M.: User stories applied: For agile software development. Addison-Wesley Professional (2004)
16. DOLSTRA, E., LÖH, A., PIERRON, N.: Nixos: A purely functional linux distribution. Journal of Functional Programming **20**(5-6), 577–615 (2010). https://doi.org/10.1017/S0956796810000195
17. Enders, R., Filkorn, T., Taubner, D.: Generating bdds for symbolic model checking in ccs. Distributed Computing **6**, 155–164 (1993)
18. Ericsson: Erlang Run-Time System Application (May 2024), https://www.erlang.org/docs/25/apps/erts/erts.pdf
19. European Commision: Regulation (eu) 2016/679 of the european parliamt and of the council of 27 april 2016 on the protection of natural persons with regard to the processing of personal data and on the free movement of such data, and repealing directive 95/46/ec (general data protection regulation). Official Journal of the European Union **59**(4) (2016)
20. European Commision: Regulation (eu) 2022/868 of the european parliament and of the council of 30 may 2022 on european data governance and amending regulation (eu) 2018/1724 (data governance act) (1 ). Official Journal of the European Union **65**(3) (2022)
21. Gheorghe, A.M., Gheorghe, I.D., Iatan, I.L.: Agile software development. Informatica Economica **24**(2) (2020)
22. Green, R., Ledgard, H.: Coding guidelines: finding the art in the science. Commun. ACM **54**(12), 57–63 (dec 2011). https://doi.org/10.1145/2043174.2043191, https://doi.org/10.1145/2043174.2043191
23. He, S., He, P., Chen, Z., Yang, T., Su, Y., Lyu, M.R.: A survey on automated log analysis for reliability engineering. ACM Comput. Surv. **54**(6) (jul 2021). https://doi.org/10.1145/3460345, https://doi.org/10.1145/3460345
24. Hoare, C.A.R.: An axiomatic basis for computer programming. Commun. ACM **12**(10), 576–580 (oct 1969). https://doi.org/10.1145/363235.363259, https://doi.org/10.1145/363235.363259

25. Hoare, C.A.R.: Communicating sequential processes. Communications of the ACM **21**(8), 666–677 (Aug 1978). https://doi.org/10.1145/359576.359585, http://dx.doi.org/10.1145/359576.359585
26. Hoder, K., Bjørner, N.: Generalized property directed reachability. In: Cimatti, A., Sebastiani, R. (eds.) Theory and Applications of Satisfiability Testing – SAT 2012. pp. 157–171. Springer Berlin Heidelberg, Berlin, Heidelberg (2012)
27. Holzmann, G.J.: The model checker spin. IEEE Transactions on software engineering **23**(5), 279–295 (1997)
28. Jhala, R., Vazou, N., et al.: Refinement types: A tutorial. Foundations and Trends® in Programming Languages **6**(3–4), 159–317 (2021)
29. King, J.: Symbolic execution and program testing. Commun. ACM **19**(7), 385–394 (Jul 1976). https://doi.org/10.1145/360248.360252
30. Lucassen, G., Dalpiaz, F., Werf, J.M.E.v.d., Brinkkemper, S.: The use and effectiveness of user stories in practice. In: Requirements Engineering: Foundation for Software Quality: 22nd International Working Conference, REFSQ 2016, Gothenburg, Sweden, March 14-17, 2016, Proceedings 22. pp. 205–222. Springer (2016)
31. Lyde, S., Might, M.: Extracting Hybrid Automata from Control Code, pp. 447–452. Springer Berlin Heidelberg, Berlin, Heidelberg; Berlin Heidelberg (2013). https://doi.org/10.1007/978-3-642-38088-4_32
32. McCormick, J.W., Chapin, P.C.: Building High Integrity Applications with SPARK. Cambridge University Press (Aug 2015). https://doi.org/10.1017/cbo9781139629294
33. Milner, R.: A calculus of communicating systems. Springer (1980)
34. Moller, F., Tofts, C.: A temporal calculus of communicating systems. In: International Conference on Concurrency Theory. pp. 401–415. Springer (1990)
35. Mues, M., Howar, F.: Gdart: An ensemble of tools for dynamic symbolic execution on the java virtual machine (competition contribution). In: International Conference on Tools and Algorithms for the Construction and Analysis of Systems. pp. 435–439. Springer (2022)
36. Norman, G., Palamidessi, C., Parker, D., Wu, P.: Model checking the probabilistic pi-calculus. In: Fourth International Conference on the Quantitative Evaluation of Systems (QEST 2007). pp. 169–178. IEEE (2007)
37. Pierce, B.C.: Types and programming languages. MIT press (2002)
38. Raharjana, I.K., Siahaan, D., Fatichah, C.: User stories and natural language processing: A systematic literature review. IEEE access **9**, 53811–53826 (2021)
39. Roscoe, B.: Model-checking csp (1994)
40. Rouhani, S., Deters, R.: Data trust framework using blockchain technology and adaptive transaction validation. IEEE Access **9**, 90379–90391 (2021). https://doi.org/10.1109/access.2021.3091327
41. Sangiorgi, D., Walker, D.: The pi-calculus: a Theory of Mobile Processes. Cambridge university press (2003)
42. Sun, J., Liu, Y., Dong, J.S.: Model checking csp revisited: Introducing a process analysis toolkit. In: International symposium on leveraging applications of formal methods, verification and validation. pp. 307–322. Springer (2008)
43. Thesing, T., Feldmann, C., Burchardt, M.: Agile versus waterfall project management: decision model for selecting the appropriate approach to a project. Procedia Computer Science **181**, 746–756 (2021)
44. Vazou, N.: Liquid Haskell: Haskell as a theorem prover. University of California, San Diego (2016)
45. Wadler, P.: Linear types can change the world! In: Programming concepts and methods. vol. 3, p. 5. Citeseer (1990)
46. Wadler, P., Kokke, W., Siek, J.G.: Programming Language Foundations in Agda (Aug 2022), https://plfa.inf.ed.ac.uk/22.08/
47. Yu, Y., Manolios, P., Lamport, L.: Model checking tla+ specifications. In: Advanced Research Working Conference on Correct Hardware Design and Verification Methods. pp. 54–66. Springer (1999)


# A The TLA$^+$ Specification

```
---------------------------- MODULE communication ----------------------------

VARIABLE DTState, clientState, inboxDT, inboxClient
TypeInvariant ≜ DTState ∈ {0, 1, 2}
                ∧ clientState ∈ {0, 1, 2}
                ∧ inboxDT ∈ {0, 1, 2}
                ∧ inboxClient ∈ {0, 1}

Init ≜ DTState = 0
       ∧ clientState = 0
       ∧ inboxDT = 0
       ∧ inboxClient = 0

SendToDT(msg) ≜ inboxDT = 0
                ∧ inboxDT′ = msg

SendToClient(msg) ≜ inboxClient = 0
                    ∧ inboxClient′ = msg

ReadFromDT(msg) ≜ inboxDT = msg
                  ∧ inboxDT′ = 0

ReadFromClient(msg) ≜ inboxClient = msg
                      ∧ inboxClient′ = 0


SendRequestToDT ≜ clientState = 0
                  ∧ SendToDT(1)
                  ∧ clientState′ = 1
                  ∧ UNCHANGED ⟨inboxClient, DTState⟩

ReadRequestFromDT ≜ DTState = 0
                    ∧ ReadFromDT(1)
                    ∧ DTState′ = 1
                    ∧ UNCHANGED ⟨inboxClient, clientState⟩

SendResponseToClient ≜ DTState = 1
                       ∧ SendToClient(1)
                       ∧ DTState′ = 2
                       ∧ UNCHANGED ⟨inboxDT, clientState⟩

ReadResponseFromClient ≜ clientState = 1
                         ∧ ReadFromClient(1)
                         ∧ clientState′ = 2
                         ∧ UNCHANGED ⟨inboxDT, DTState⟩
```

```
SendAckToDT ≜ clientState = 2
               ∧ SendToDT(2)
               ∧ clientState′ = 0
               ∧ UNCHANGED ⟨inboxClient, DTState⟩

ReadAckFromDT ≜ DTState = 2
                 ∧ ReadFromDT(2)
                 ∧ DTState′ = 0
                 ∧ UNCHANGED ⟨inboxClient, clientState⟩

Next ≜ SendRequestToDT
    ∨ ReadRequestFromDT
    ∨ SendResponseToClient
    ∨ ReadResponseFromClient
    ∨ SendAckToDT
    ∨ ReadAckFromDT

Spec ≜ Init ∧ □[Next]_⟨DTState, clientState, inboxDT, inboxClient⟩

────────────────────────────────────────────────────────────

THEOREM Spec ⇒ □TypeInvariant

════════════════════════════════════════════════════════════
\* Modification History
\* Last modified Tue Jul 16 11:34:25 CEST 2024 by haetze
\* Created Wed Jul 10 08:18:17 CEST 2024 by haetze
```