Es gibt zwei Voraussetzung für eine Public-Key-Infrastruktur:

1.

Der private Schlüssel darf nur dem Teilnehmer bekannt sein, der den Schlüssel besitzt.

Der private Schlüssel darf also nicht publiziert oder entwendet werden.

2.

In einem Netzwerk muss der öffentliche Schlüssel sicher der Identität eines Teilnehmers zugeordnet sein.

Dies kann durch bekannte Handshake-Verfahren sichergestellt werden.

Sind beide Voraussetzungen erfüllt, können Daten mit Hilfe des privaten Schlüssels signiert werden.

Die Signatur kann mit Hilfe des öffentlichen Schlüssels validiert werden.

Ausgetauschte Daten sind dadurch vor Manipulation geschützt.

Außerdem lässt sich die Herkunft der ausgetauschten Daten eindeutig identifizieren.

Auf diesem Verfahren beruht Code-Signing und die digitale Unterschrift auf Dokumenten.

Im Folgenden wird eine Erweiterung des Verfahrens beschrieben.

Die Erweiterung bezieht sich auf drei Aspekte:

1. Signierung des Zustands von Anwendungen, die durch ein Netzwerk verteilter Teilnehmerknoten ausgeführt werden

2. Signierung des Codes derartig verteilter Anwendungen

3. Reproduktion des Anwendungszustands

Das Verteilen der Kombination aus Code und Zustand auf einzelne Teilnehmer

kommt einem Kontrakt

Dadurch ist es Teilnehmerknoten möglich, Konsens bezüglich Code und Anwendungszustand zu festzustellen.

Kann Dissens auch durch das Konsultieren von weiteren Knoten nicht ausgeräumt werden, brechen die Teilnehmerknoten die Verarbeitung ab.

Alle Knoten verfügen über eine Implementierung der gleichen Laufzeitumgebung.

Der Status der Anwendung wird im xml-Format gespeichert.

Teilnehmende Knoten werden im XML in einer Struktur des Typs "Participant" bestehend aus den Feldern Nickname, Mailadresse und öffentlichem Schlüssel beschrieben.

Die Rolle der teilnehmenden Knoten wird im XML in einer Struktur des Typs "Duty" festgelegt. Duty enthält Code, der von der Laufzeitumgebung des teilnehmenden Knoten ausgeführt wird. Entsprechend wird Duty individuell implementiert.

Empfängt ein Knoten ein XML, überprüft er zunächst ob er zu Recht als "Participant" aufgeführt wird. Falls ja, hat der Knoten festgestellt, dass er ein teilnehmender Knoten ist. Er durchsucht das XML nach den ihm zugewiesenen "Duties" und führt diese nacheinander aus (von oben nach unten).

CODE

Der auszuführende Code liegt in einer Programmiersprache vor, die folgende Eigenschaften hat:

- statische Typisierung

- Primitive Datentypen

- numerische Datentypen sind ausschließlich ganzzahlig (Integer, Long, BigInteger)

- Der einzige Zeichen-Typ ist ASCII. Die maximale Länge von Strings wird bei der Typ-Definition angegeben. Unicode-Strings müssen als Base64 codiert werden.

- Der Boolsche Datentyp wird wird im XML durch die Strings "true" und "false" codiert.

- Instanzen primitiver Datentypen werden explizit durch eine entsprechende Anweisung erzeugt

- Für die numerischen Datentypen stehen die elementare Arithmetik (+, -, \*, /, mod) zur Verfügung

- Innerhalb der arithmetischen Operation können die verschiedenen numerischen Datentypen benutzt werden. Numerischer Überlauf führt zum Abbruch der gesamten Verarbeitung. Die Laufzeitumgebung liefert eine Fehlermeldung.

- Die arithmetischen Operationen können verschachtelt werden.

- Für den Zeichen-Typ stehen die Operationen search, substring, concat zur Verfügung

- Für alle primitiven Datentypen stehen die Vergleichs-Operationen =, <, > zur Verfügung

- Für die Vergleichs-Operationen und den boolschen Datentyp stehen die boolschen Operationen "Not", "And", "Or", "XOr" zur Verfügung

- Die boolschen Operationen können verschachtelt werden.

- Für den Kontrollfluss stehen if-else, sowie die while-Schleife zur Verfügung

- Complexe Datentypen sind Kombinationen aus Feldern primitiver oder complexer Typen

- Instanzen complexer Datentypen werden explizit durch eine entsprechende Anweisung erzeugt

- Zuweisungen unter Feldern primitiven Typs sind nur unter Übereinstimmendem Basis-Datentyp (Numerisch, String, Boolean)

- Complexe Datentypen können aus einem Supertyp abgeleitet werden (einfache Vereerbung)

- Zuweisungen unter Feldern complexen Typs sind nur möglich wenn der Typ des Zielfeld dem Typ des Quell-Feldes entspricht oder wenn der Typ des Quell-Feldes vom Typ des Zielfeldes erbt.

- Es können Funktionen innerhalb von Complexen Datentypen definiert werden.

- Funktionen können Rückgabewerte haben (Commando "return")

- Gemäß Datentyp können Funktionsaufrufe innerhalb von arithmetischen, Vergleichs- oder boolschen Operationen benutzt werden.

- Rekursion ist nicht möglich

- Nebenläufigkeit, Multithreading ist nicht möglich

Jede weitere Funktionalität darf nur dann in den Sprachumfang aufgenommen werden, wenn sichergestellt werden kann, dass jeder Teilnehmerknoten bei der Reproduktion des Anwendungszustands zum gleichen Ergebnis kommt.

Entsprechend ist kein Zugriff auf die individuellen Systemresourcen (Filesystem, Uhrzeit, Gleitkomma-Einheiten) vorgesehen.

Enthält der Code eine Zuweisung auf ein Feld, dass über die Wurzel-Struktur "Contract" adressierbar ist, so kommt dies der Änderung des Anwendungszustands gleich.

Das geänderte XML wird daher neu signiert. Die Signatur wird innerhalb des XMLS in einer Struktur zusammen mit Nickname, Mailadresse und öffentlichem Schlüssel gespeichert. Der öffentliche Schlüssel wird in Base64 kodiert.

Der als XML vorliegende signierte Status kann nun unter teilnehmenden Knoten ausgetauscht werden. Für den Austausch sind diverse Protokolle (http, FTP, Socket, Mail, File-System ...) vorgesehen.

Nur so ist es möglich, dass alle Knoten über eine Implementierung der gleichen Laufzeitumgebung verfügen und Konsens bezüglich des Anwendungszustands feststellen können.

Höhere Funktionalität wird von der Laufzeitumgebung lediglich abstrakt zur Verfügung gestellt. Der Code der Duty fordert diese Funktionalität mit dem Commando "Quest" an und erhält von der Laufzeitumgebung einen Rückgabewert primitiven Typs. Die Laufzeitumgebung beanwortet Quests mithilfe von Plugins, die für bestimmte Quest registriert werden können. Wir bezeichnen derartige Plugins als Agent. Im einfachsten Fall ist der Agent eine GUI, die einem menschlichen Betreiber des Knotens ermöglicht, Daten manuell einzugeben. Der Agent GUI ist der Default-Agent, der Quests beantwortet, auf die kein Agent registriert wurde. Seitens der Laufzeitumgebung ist damit sichergestellt, dass jeder Quest beantwortet werden kann.

Um eine Reproduktion des Anwendungszustands durch andere Knoten zu ermöglichen, werden die Rückgabewerte zu jedem Quest in einer Stack-Struktur innerhalb des XMLs gespeichert und stehen dadurch dem Validierer zur Verfügung.

Jeder Contract muss das Verhalten bei Dissens explizit selbst implementieren, ansonsten verbleibt der Contract in einem Zustand, der von keinem teilnehmenden Knoten weiterverarbeitet wird. Eine denkbare Implementierung für Dissens wäre, die Entscheidung über den tatsächlichen Zustand von einem dritten teilnehmenden Knoten vornehmen zu lassen. Dieser Knoten würde dann sozusagen als Boss oder Richter fungieren.

Ein Contract bildet den Zustand eines konkreten zu verarbeitenden Falls ab. Dies kann der operativen Umsetzung einer Vereinbarung auf höherer Ebene entsprechen. Ein Beispiel für die Vereinbarung auf höherer Ebene ist ein SLA, der festlegt, welcher Sachbearbeiter welche Klasse von Anfragen bearbeitet. Der Contract, der den Bearbeitungsstand einer konkreten Anfrage abbildet würde auf diesen SLA-Contract referenzieren, so dass jeder teilnehmende Knoten überprüfen kann, ob ihm Duties zu Recht zugewiesen wurden.

Durch das Referenzieren anderer Contracts ist es möglich, Protokolle umzusetzen, die als Basis des Contract dienen.

Themenspeicher

Dadurch, dass der Code zur Laufzeit kompiliert wird sind Änderung des Codes zur Laufzeit möglich.