Hybrid Edge Computing

Enable Processing of Sensitive Data in Mobile Edge Computing Scenarios

Andreas Reiter

Dipl.-Ing. Andreas Reiter Bakk.techn.

Hybrid Edge Computing

Enable Processing of Sensitive Data in Mobile Edge Computing Scenarios

DOCTORAL THESIS

to achieve the university degree of

Doktor der technischen Wissenschaften

submitted to

Graz University of Technology

Supervisor

O.Univ.-Prof. Dipl.-Ing. Dr.techn. Reinhard Posch

Institute of Applied Information Processing and Communications
Faculty of Computer Science and Biomedical Engineering

Graz, September 2017

Dipl.-Ing. Andreas Reiter Bakk.techn.

Hybrid Edge Computing

Enable Processing of Sensitive Data in Mobile Edge Computing Scenarios

DISSERTATION

zur Erlangung des akademischen Grades

Doktor der technischen Wissenschaften

eingereicht an der

Technischen Universität Graz

Betreuer

O.Univ.-Prof. Dipl.-Ing. Dr.techn. Reinhard Posch

Institut für Angewandte Informationsverarbeitung und Kommunikationstechnologie
Fakultät für Informatik und Biomedizinische Technik

Graz, September 2017

Statutory Declaration

I declare that I have authored this thesis independently, that I have not used other than the declared sources / resources, and that I have explicitly indicated all material which has been quoted either literally or by content from the sources used.

Eidesstattliche Erklärung

Ich erkläre an Eides statt, dass ich die vorliegende Arbeit selbstständig verfasst, andere als die angegebenen Quellen/Hilfsmittel nicht benutzt, und die den benutzten Quellen wörtlich und inhaltlich entnommenen Stellen als solche kenntlich gemacht habe.

SIGNATURINFORMATIONEN	**Unterzeichner**	Andreas Reiter
	Datum/Zeit-UTC	2017-09-07T16:18:23+02:00
	Prüfinformation	Informationen zur Prüfung der elektronischen Signatur finden Sie unter: https://www.signaturpruefung.gv.at
Hinweis	Dieses mit einer qualifizierten elektronischen Signatur versehene Dokument hat gemäß Art. 25 Abs. 2 der Verordnung (EU) Nr. 910/2014 vom 23. Juli 2014 ("eIDAS-VO") die gleiche Rechtswirkung wie ein handschriftlich unterschriebenes Dokument.	

Abstract

In recent years, mobile-device distribution was skyrocketing, even with multiple devices per user. Mobile devices are equipped with increasingly powerful hardware components, also enabling execution of computationally intensive applications. However, it turns out that battery development cannot keep up with the developments of the semiconductor industry. This leaves users in the situation that they need to recharge their devices multiple times a day if utilising the devices' full potentials.

It is a well accepted and proven fact that offloading computations can help in saving energy and even have positive impacts on the overall performance. Existing, well-established solutions to this problem force mobile devices and applications into rigid structures without considering or reacting to their environmental parameters. State-of-the-art solutions at least try to add dynamic mechanisms on the application side but remain static on the server side. Furthermore, it turns out that data security and protecting users' privacy only play minor roles in existing approaches. Therefore, whole groups of users, with strict security requirements are unable to make use of offloading technologies.

This thesis removes these restrictions. A threat model serves as a solid foundation to identify security gaps and misconceptions in existing offloading frameworks and approaches. The first impression that existing frameworks do not consider data security aspects can be confirmed. Currently established models are unable to provide the mandatory security properties. The findings of these research results in the main contribution of this thesis - a novel mobile-first processing model: Hybrid Mobile Edge Computing (HMEC). The model breaks with centralised structures and enables offloading to different devices, using different communication technologies, without relying on single management endpoints. It enables a continuous operation and high degree of fail-safety. Still, the data controllers remain in full control of their data.

The security concepts of HMEC are built on solid foundations. The author of this thesis was in charge of leading research and development of a data security governance model targeted at federated cloud environments as part of the EU SUNFISH project. Due to structural and functional commonalities of federated cloud environments and mobile cloud computing processing models, it was self-evident to transform relevant parts of the governance architecture to the mobile world. Therefore, the overall HMEC processing model as well as the security concepts were validated by broadly exposing and presenting them to relevant scientific communities. The transformation and adaptation of federated cloud computing concepts allows to reuse already existing infrastructures and knowledge, XACML as a de-facto standard in data security policy languages is just one example. The proposed models and implementations are backed by thorough evaluations targeting the energy- and performance-saving effects as well as security aspects.

Summarising, the HMEC processing model enables computational offloading where sensitive data is involved. Still, data controllers remain in full control of their data and involved workflows. This solid security foundation qualifies the model to act as a baseline for future developments in the sector of mobile cloud computing.

Kurzfassung

Aktuelle Entwicklungen im Bereich von Mobile Computing werden von der immer weiteren Verbreitung von mobilen Geräten vorangetrieben. Mobile Geräte werden von Jahr zu Jahr leistungsfähiger und sind mittlerweile sogar in der Lage Applikationen mit hohen CPU Anforderungen auszuführen. Es hat sich jedoch gezeigt, dass die Entwicklungen am Akkusektor langsamer voranschreiten, als die Entwicklung der restlichen Hardware. Unter Volllast hat die Hardware einen steigenden Energiebedarf, der von den verbauten Energiequellen nicht bereitgestellt werden kann. Dies versetzt die Benutzer in die Lage, dass ihre mobilen Geräte mehrfach täglich neu geladen werden müssen, wenn die gesamte Palette der verfügbaren Funktionalität ausgereizt wird.

Um dem entgegen zu wirken wurde das Computational Offloading entwickelt, das dazu beiträgt Energie zu sparen und sogar die Performance verbessert. Bestehende Systeme wenden allerdings sehr starre Lösungsansätze an, ohne Umgebungsparameter miteinzubeziehen. Der letzte Stand der Technik ermöglicht zumindest Clientseitige dynamische Entscheidungen, führt diesen Ansatz auf der Gegenseite aber nicht fort. Das größte Manko zeichnet sich aber im Bereich der Datensicherheit und des Schutzes der Privatsphäre von Benutzern ab. Diese Aspekte spielen derzeit nur eine untergeordnete Rolle. Dieses Vorgehen schließt ganze Benutzergruppen, die großen Wert auf Datensicherheit legen, aus.

Das Ziel dieser Arbeit ist es, die aktuell sehr limitierende Situation erheblich zu verbessen. Als Ausgangspunkt fungiert eine fundierte Bedrohungsanalyse mit dem Fokus auf die Mobile Computing Domäne um Schwachstellen in bestehenden Systemen aufzudecken. Die anfänglichen Befürchtungen, dass bestehende Lösungen Aspekte der Datensicherheit nicht berücksichtigen wurden dadurch bestätigt. Bestehende Ansätze sind nicht in der Lage die gewünschten sicherheitsrelevanten Eigenschaften bereitzustellen. Dies resultiert im Hauptbeitrag dieser Arbeit - ein neuartiges Ausführungsmodell, ausgerichtet auf mobile Applikationen: Hybrid Mobile Edge Computing (HMEC). In diesem Modell werden Bezüge zu zentralen Komponenten vermieden. Aufgaben können auf verschiedenste Geräte unter Verwendung von unterschiedlichen Kommunikationstechnologien ausgelagert werden, ohne dabei auf zentrale Verwaltungskomponenten angewiesen zu sein. Dies ermöglicht einen durchgehenden Betrieb und erhöht die Fehlersicherheit erheblich.

Die Sicherheitskonzepte des HMEC Modells bauen auf einer soliden Basis auf. Der Verfasser dieser Arbeit war federführend bei Forschung und Entwicklung eines "Data Security Governance Models" für föderierte Cloudumgebungen als Teil des EU SUNFISH Projekts. Durch strukturelle und funktionelle Gemeinsamkeiten der föderierten Cloudumgebung und des HMEC Modells, war es naheliegend relevante Teile des föderierten Cloudansatzen in die mobile Welt zu transferieren. Die solide Basis, aber auch das transferierte Modell, wurde daher einer breiten wissenschaftlichen Diskussion unterzogen. Außerdem erlaubt die Transformation des Modells die Wiederverwendbarkeit von bereits bestehenden Infrastrukturen, sowie von bereits breit verfügbarem Wissen. XACML, als quasi Standard bezüglich Data Security Policy Languages ist hierbei nur ein Beispiel. Die entwickelten Modelle und Umsetzungen werden durch eine gründliche Evaluierung bezüglich Verbesserungen im Energieverbrauch, Performancesteigerungen, aber auch mit Blick auf Sicherheitsaspekte untermauert.

Das HMEC Modell, als Resultat dieser Arbeit, erlaubt es somit rechenintensive Aufgaben auszulagern, auch wenn sensible Daten involviert sind. Die Datenverantwortlichen behalten die volle Kontrolle über ihre Daten, sowie die involvierten Workflows. Die solide Basis des Modells, auch in Bezug auf Sicherheitsaspekte, qualifiziert das Modell um auch für zukünftige Entwicklungen im Mobile Cloud Computing Bereich als solider Unterbau herangezogen zu werden.

Contents

List of Figures

List of Tables

List of Listings

List of Acronyms

ABAC Attribute-based Access Control

AFS Andrew File System

BYOD Bring Your Own Device

CC Cloud Computing

DFS Distributed File System

DHT Distributed Hash Table

DTS Data Transformation Services

ETSI European Telecommunications Standards Institute

GDPR General Data Protection Regulation

GPFS General Parallel File System

HMCC Hybrid Mobile Cloud Computing

HMEC Hybrid Mobile Edge Computing

IaaS Infrastructure-as-a-Service

ICE Interactive Connectivity Establishment

IPC Inter-process Communication

JSON JavaScript Object Notation

JWT JSON Web Token

LTE Long Term Evolution

MAC Message Authentication Code

MCC Mobile Cloud Computing

MDM Mobile Device Management

MEC Mobile Edge Computing

NAT Network Address Translation

NFS Network File System

NIST National Institute of Standards and Technology

P2P Peer-to-Peer

PaaS Platform-as-a-Service

PAP Policy Administration Point

PDP Policy Decision Point

PEP Policy Enforcement Point

PIP Policy Information Point

PRP Policy Retrieval Point

QoS Quality of Service

RAN Radio Access Network

RBAC Role-based Access Control

RTT Round-trip-time

SaaS Software-as-a-Service

SGX Software Guard Extensions

STUN Session Traversal Utilities for NAT

TLS Transport Layer Security

TURN Traversal Using Relay NAT

UMTS Universal Mobile Telecommunication System

WebRTC Web Real-Time Communication

XACML Extensible Access Control Markup Language

XML Extensible Markup Language

Part I

Problem Definition

Chapter 1

Introduction

Decades ago, IT outsourcing was big business [J.-n. Lee et al., 2003]. In the early days, outsourcing was realised using time-sharing and processing services. At this time computers were large and maintenance intensive. Therefore, the operation was left to service bureaus, system houses, or other professional firms. With the introduction of affordable personal computers, the outsourcing business vanished and shifted to vertical integration approaches. Since then, IT was considered an in-house function. Each organisation operated their information systems based on standard equipment available in the market but assembled to customised environments. Later, outsourcing was revitalised, but not for contract programming or specific services. The outsourcing industry shifted to system integrations including complex technologies like network management and communications, covering the whole operation of on-site facilities including IT personnel. These outsourcing technologies evolved over the recent years and resulted in completely outsourced corporations and environments with the ability to scale their computing resources dynamically according to the current needs.

These approaches and technologies are focussed on corporations and service providers to offer seamless operations and scalability for services. The users' needs are only considered second. Especially with massive mobile device usage, feature rich and computationally intensive mobile applications, users demand long battery lifetime and seamless operation, regardless of their current environment. This requires dynamically adapting approaches, which makes use of different computing resource types.

This thesis develops approaches to achieve the same advantages as corporations have today to outsource their IT infrastructure and services but for mobile device users. New mobile application execution models are developed and evaluated with a strong focus on data security. In this chapter, the motivation and methodology followed by this thesis are described, as well as a detailed view on the contributions and structure of the document is provided.

1.1 Motivation

In the recent years, the usage of mobile devices, such as smartphones and tablets, outperformed the usage of desktop computers. Classic desktop computer applications are migrated to mobile device applications with heavy impacts on user experience and data security. Today, two major approaches are used to migrate applications.

1. *On-device execution:* Applications are developed with a majority of their application logic in the application package, executed on the device. Certainly, applications will use external services and will not operate isolated, but processing the input data is performed on the mobile device.

2. *Cloud (-assisted) execution:* Cloud computing acts as an enabling technology in the sector of mobile computing. In cloud-assisted scenarios, applications are separated into parts executed in

the cloud and parts executed locally. Well-defined interfaces enable a seamless interaction between those parts. A practical approach is to execute computationally intensive application parts in the cloud and even increase the user experience on the mobile device, under optimal conditions. Pushing this approach forward can give cloud computing even more responsibilities. In scenarios where the cloud executes the whole application, mobile devices only act as thin clients displaying the interfaces to the user and providing the relevant user input.

Both ways are obviously valid approaches to provide applications on mobile devices. Their practicability depends on the application characteristics. The increasing available computational power of mobile devices even enables computationally intensive applications to be executed on mobile devices with the downside of decreasing battery lifetime. Coughlin [2015] provides an analysis of currently prevalent energy sources for mobile devices and their evolution. The semiconductor's price-performance ratio still sticks to Moore's law, battery performance and price, in contrast, does not improve at the same rate. According to his analysis and estimations [Coughlin, 2015], energy density has been doubling every ten years. Transistor density, in contrast, has been roughly doubling every 18 months. This situation results in a clear gap between application migration to mobile devices with high computational and energy demands, and the available battery technologies. The second approach, cloud-assisted execution, eliminates these issue by outsourcing computations to cloud computing resources. This approach works great under optimal environmental conditions but fails or decreases user experience in cases like:

- Low bandwidth connection conditions to the cloud computing resource.
- High latencies due to bad Internet connectivity.
- High roaming costs.

Existing approaches and corresponding frameworks provide valuable contributions but mostly follow a static approach. They exclusively link devices to single computing resources, giving the mobile device no chance to dynamically react to changing environments. Furthermore, the practicability is often hindered by lock-ins to specific platforms, operating systems, or operating system versions. More flexible ways are required to provide the optimal experience for users and maintain their battery lifetime. Due to their static nature, existing approaches and frameworks only considered data security second and limit their contributions to pre-established trust relationships. Flexible systems are subject to greater attack surfaces, and pre-establishing trust relationship is not a viable solution. Dynamic approaches are required, also regarding data security to protect users' data and computations.

1.2 Methodology

The foundation for yielding novel research results is a deep understanding of the state-of-the-art as well as gaining awareness of research gaps. Concerning mobile cloud computing and related fields of research, lots of valuable contributions are available. These researches mostly concern client-side techniques for application analysis. The absence of data security considerations and dynamic approaches is striking. To substantiate this observation a threat model is developed, covering the peculiarities of dynamically executed mobile applications. The threat model acts as the foundation for the definition of security requirements. It follows a well-established threat modelling approach defining the target of evaluation with corresponding assumptions and identifying assets. Threat agents and threats are identified which put the assets in danger. The result is a security requirement catalogue, which is evaluated against a representative set of existing frameworks. This analysis confirms the first impression that existing solutions do not focus on security related aspects. The catalogue enables to identify gaps in existing architectures regarding security-awareness, also on a functional level.

The identified gaps and an abstract architecture of current approaches serve as the baseline for research activities to improve current models, closing the gaps and already considering data security on the architectural level. The result of this research is a novel mobile-focused processing model: Hybrid Mobile Edge Computing (HMEC). Using the security requirement catalogue as a foundation, data security plays a major role during research on the HMEC processing model. The utilised systematic approach, starting from analysing existing frameworks, identifying the gaps and creating a security requirements catalogue, yields a thorough data security model. The potentials of the processing model are highlighted and backed by a full-featured implementation, as well as corresponding evaluations. Technologies are identified which enable a security-aware and efficient implementation. The author of this thesis was in charge of leading research on a data security governance model for federated cloud environments in a European project setup. This model is transferred to the mobile world and tightly integrated with the HMEC processing model.

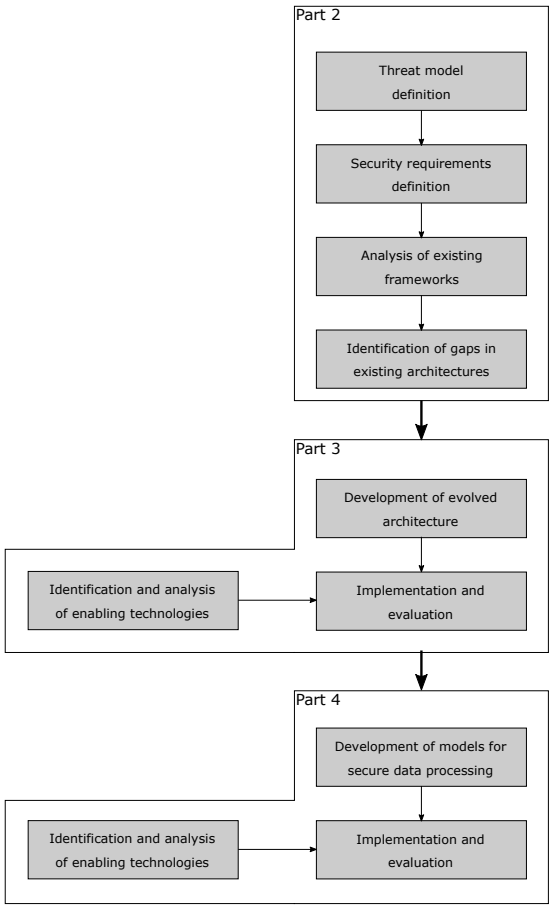


Figure 1.1: Overview of the followed methodology

Figure 1.1 shows the structured process model and the dependencies between the performed actions. Furthermore, the actions are arranged according to their affiliation to the parts in this thesis. The illustration shows the abstract goal of this thesis: evolve current mobile cloud computing approaches and enable a more dynamic usage, still increasing security-awareness to enable operation on sensitive data, if required.

1.3 Contribution

The research on the topics related to this thesis can be separated into two large clusters. The first cluster considers mobile application execution models. A security assessment framework is developed, and existing approaches and frameworks are evaluated [Reiter and Zefferer, 2015a]. The identified flaws pave the way and influence the development of new approaches and architectures in this regard [Reiter and Zefferer, 2015b; Reiter and Zefferer, 2016]. Enabling technologies and possible extensions are scrutinised in [Reiter, 2015]. The security and trust related parts have a strong connection to the European H2020 SUNFISH project. In this project, the current state-of-the-art regarding cloud federations is analysed, focusing on security aspects. Based on this analysis an information sharing governance model is developed [Suzic and Reiter, 2016]. The findings of this model are successfully transformed from the domain of federated clouds to the domain of mobile application execution in distributed and federated environments [Reiter, 2017]. An additional security and impact analysis are performed for enabling technologies [Reiter and Marsalek, 2017]. A complete view of the problem description together with a sound solution, considering all relevant aspects from both clusters was published in [Reiter, Prünster, and Zefferer, 2017].

In the following, a listing of relevant publications in chronological order is provided, all with substantial contribution to the topic:

Andreas Reiter [2015]. "Enabling Secure Communication over Existing Peer-to-Peer Frameworks". *2015 23rd Euromicro International Conference on Parallel, Distributed, and Network-Based Processing* (2015), pages 575–582. ISSN 1066-6192. doi:10.1109/PDP.2015.10. http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=7092777

[Reiter, 2015]

Andreas Reiter and Thomas Zefferer [2015a]. "Paving the Way for Security in Cloud-Based Mobile Augmentation Systems". In: *3rd IEEE International Conference on Mobile Cloud Computing, Services, and Engineering (IEEE Mobile Cloud 2015)*. 2015, pages 89–98. ISBN 978-1-4799-8977-5. doi:10.1109/MobileCloud.2015.12. http://ieeexplore.ieee.org/xpl/articleDetails.jsp?arnumber=7130873

[Reiter and Zefferer, 2015a]

Andreas Reiter and Thomas Zefferer [2015b]. "POWER: A cloud-based mobile augmentation approach for web- and cross-platform applications". *2015 IEEE 4th International Conference on Cloud Networking, CloudNet 2015* (2015), pages 226–231. doi:10.1109/CloudNet.2015.7335313

[Reiter and Zefferer, 2015b]

Andreas Reiter and Thomas Zefferer [2016]. "Flexible and Secure Resource Sharing for Mobile Augmentation Systems". *2016 4th IEEE International Conference on Mobile Cloud Computing, Services, and Engineering (MobileCloud)* (2016), pages 31–40. doi:10.1109/MobileCloud.2016.8. http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=7474403

[Reiter and Zefferer, 2016]

Bojan Suzic and Andreas Reiter [2016]. "Towards Secure Collaboration in Federated Cloud Environments". *Workshop on Security, Privacy, and Identity Management in the Cloud* (2016)

[Suzic and Reiter, 2016]

Andreas Reiter [2017]. "Secure Policy-based Device-to-Device Offloading for Mobile Applications". In: *32nd ACM Symposium on Applied Computing*. In Press, 2017. ISBN 9781450344869

[Reiter, 2017]

Andreas Reiter and Alexander Marsalek [2017]. "WebRTC: Your Privacy is at Risk". *32nd ACM Symposium on Applied Computing* In press (2017)

[Reiter and Marsalek, 2017]

Andreas Reiter, Bernd Prünster, and Thomas Zefferer [2017]. "Hybrid Mobile Edge Computing: Unleashing the Full Potential of Edge Computing in Mobile Device Use Cases". In: *2017 17th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGrid)*. In Press, 2017

[Reiter, Prünster, and Zefferer, 2017]

1.4 Outline

This thesis is structured according to the defined methodology and follows a bottom-up approach. The thesis is structured in five major parts. Each part provides a considerable contribution, together forming a sound picture.

Part I settles the scene for this thesis providing an overall problem description. Furthermore, relevant baseline approaches, technologies, and mobile application execution models are described. On a technological level, grid computing is described as an enabling baseline technology and its development to today's prevalent cloud computing. Cloud computing is highlighted from a user's point-of-view, focusing on the service aspects of cloud computing. Additionally, a novel technology, mobile edge computing, is introduced, targeted at the needs of today's mobile device users. Besides the technological foundation, mobile application execution models targeted at a flexible and dynamic execution of mobile execution are considered and their weaknesses are highlighted. The first part is concluded with the proposal of an extended execution model, focussed on mobile device users and taking into account all available baseline technologies. Part I highlights the needs to research in the direction of dynamic and flexible application execution.

Part II of this thesis contributes by analysing existing frameworks and developing a security assessment catalogue to assess the handling of sensitive data and computations of frameworks. The security assessment catalogue is the result of a threat modelling process according to well-established and accepted means. The resulting catalogue is then assessed against existing frameworks and essentially influences the further architectural development.

Part III continues on an architectural level. First, a common architecture is derived from existing frameworks and weaknesses are identified. A new improved architecture specification is derived, considering the weaknesses identified up to this point. The conducted performance and energy impact analyses, based on a full featured implementation, show the potentials of this approach.

Part IV completely focusses on trust, security and privacy issues. A governance approach is defined to protect users' data and computations. On a technical level, assessment approaches are analysed to assess the trustworthiness of remote computing units. Furthermore, their integration strategies are discussed.

Part V provides other, relevant and required background knowledge. The outcome of this thesis is a new model to execute mobile applications in an energy saving and performance enhancing way.

Chapter 2

Cloud Computing and Related Concepts

Cloud computing is subject to various definitions, the overall goal of this chapter is to settle a solid definition, valid in the context of this thesis. Nowadays cloud computing is used as a buzzword to illustrate properties like endless scalability regarding processing power and memory, and a service oriented behaviour without focussing on the details. Providers introduce cloud computing as a novel innovation but do not mention the technology's roots.

In this chapter, the baseline technology of cloud computing is introduced in Section 2.1, and the evolution of today's cloud computing. Following in Section 2.2, a common definition of cloud computing is introduced and their usages in different fields, including backends for mobile applications. In Section 2.3 the next evolutionary step of cloud-supported applications is introduced. Mobile edge computing is particularly focused on mobile devices and enables incorporation of the user's context.

2.1 Grid Computing

Grid computing means different things to different entities, therefore a variety of definitions exists in literature. This thesis uses the definition from Jacob et al. [2005] as its baseline but brings it into relation with other concepts.

Grid computing can be considered as one solution to solve problems in a distributed manner. The term *Grid* can be traced back to the initial idea of organising computers in a power grid-like structure. Users just have to plug into their wall sockets and use the provided functionality, without agonising about the provisioning or other details. Although this initial vision has never been commercially realised in its full extent in the domain of grid computing, it enables to identify the technological foundation. In the end, the key values of grid computing are in the underlying distributed computing technologies to enable a cross-organisational resource sharing. This is enabled by using virtualisation across platforms and architectures by using open standards. These technologies facilitate virtualisation at different levels: Starting from single systems, over storage and network, to whole enterprises. Hence, a grid is a collection of entities contributing their resources of different types, waiting for jobs to execute. The contributed resources can have various types. The two most prominent are *computing cycles* and *storage*:

Computing cycles are the primary contribution. Computing power is provided to the grid, differing in architecture, speed and memory. Contributed computing power can only be utilised if a matching usage model is provided. Three fundamental approaches have emerged:

- An application is completely run in the grid on one of the offered machines.
- An application, which requires multiple runs on different datasets, is executed in parallel on various machines.

- An application offers ways to split its work into separate jobs, which are executed in parallel on different processors.

In grid computing, scalability is the measure of all things. As Jacob et al. [2005] state, the execution of an optimally scalable application only takes half the time if twice as many processors are available. In most cases, there is a logical upper limit in splitting to separate jobs.

Storage contributions are also a commonly shared grid resource type, also referred to as *data grids*. Leaving CPU caches and volatile memory aside, shared hard disk space can be used e.g. to increase capacity, performance, and reliability. These properties are driven by sophisticated distributed file systems like the Andrew File System (AFS)[1], Distributed File System (DFS)[2], Network File System (NFS) [Shepler et al., 2003], and General Parallel File System (GPFS) [Schmuck and Haskin, 2002]. They enable to span single file systems across multiple machines. Users refer to files in an ordinary way, without explicitly knowing the exact machine where the file is located. More advanced file systems e.g. automatically duplicate data at multiple sites to prevent data loss, provide increased reliability and performance.

Besides these two major resource types, others are often referred to as: communication bandwidth, software, or specific hardware. Still, computing cycles and storage are the predominant resource types and are most important for the further considerations in this thesis.

With grid computing and grid-enabled applications in place, the following benefits could manifest according to Jacob et al. [2005]:

- *Exploiting underutilised resources:* Grid computing provides a framework to provide underutilised resources to the grid and make use of other's resources in peak times. This does not only apply to computational grids, where grid-aware applications are executed on different machines, but also to data grids. Many machines have free disk space, which could be provided to others using a data grid.

- *Parallel CPU capacity:* Applications, providing the capabilities to split their computationally intensive tasks into many small tasks can ideally make use of the massive parallelism in grid computing. Barriers exist in transforming applications to grid-enabled applications. Usually, algorithms have an upper limit for splitting into separate tasks, or can only be split into dependent tasks. These limits influence the performance gains due to parallelisation in a negative way.

- *Virtual resources:* Due to the virtualisation of resources, sharing across a wide audience is achieved. It is not limited to computational resources, but also datasets, specialised hardware, or software (in a software license sharing model).

- *Resource balancing:* Grids can serve to absorb unexpected activity peaks in a load balancing manner. Even in a fully utilised grid, an absorption is possible through task prioritisation.

- *Reliability:* By scheduling the same self-contained task to multiple machines at the same time, and make a decision about the correct results, incorrect behaving nodes can be detected.

- *Management:* All the possible benefits, in essence, enable a smooth manageability of resources and paves the way to get rid of rigid resource assignments to single groups and therefore save costs.

Researchers developed countless systems following the grid computing paradigms, accessible by all their consortium members like TeraGrid [Beckman, 2005] and Open Science Grid [Pordes et al., 2007]. Domain specific grids like caBIG [Fenstermacher et al., 2005] focused on cancer research, which also makes use of the grid data sharing capabilities, Earth System Grid [Bernholdt et al., 2005] focused on

[1] http://pages.cs.wisc.edu/~remzi/OSTEP/dist-afs.pdf

[2] https://technet.microsoft.com/en-us/library/cc753479(v=ws.10).aspx

climate research and also general grid frameworks like EGEE [Laure et al., 2004] were developed. But, according to Foster et al. [2008], a commercial breakthrough was not achieved. No commercial grid providers emerged. Still, grid computing serves as the technological foundation for superior approaches.

2.2 Cloud Computing

The challenge of finding a clear definition of grid computing, continuous in the domain of cloud computing. This fact is also highlighted by the National Institute of Standards and Technology (NIST) with providing a survey over a plethora of available cloud computing reference models from different viewpoints [NIST, 2011].

From an abstract point-of-view, this thesis adheres to the definition of Foster et al. [2008]:

> A large-scale distributed computing paradigm that is driven by economies of scale, in which a pool of abstracted, virtualised, dynamically-scalable, managed computing power, storage, platforms, and services are delivered on demand to external customers over the Internet. [Foster et al., 2008]

This definition has a clear overlapping with the definition of grid computing in Chapter 2.1. Grid computing provides the necessary backbone technology. Hence, cloud computing evolved out of grid computing but shifts its focus from providing compute and storage resources to providing more abstract resources. To a certain extent, cloud computing provides a business model for the grid computing backbone, in terms of metered resources or services. Basically, grid- and cloud computing follow identical visions: "...reduce the cost of computing, increase reliability, and increase flexibility by transforming computers from something that we buy and operate ourselves to something that is operated by a third party."[Foster et al., 2008]. But cloud computing is, in contrast, extensively using virtualisation technology, with global commercial companies in place building and maintaining large-scale systems containing massive loads of computers.

Foster et al. [2008] identified key factors, differentiating cloud computing from grid computing. Regarding the business model in cloud computing, users pay on a consumption basis like consumption time of processors and amount of stored data for a given timeframe. In the available (scientific) grid computing infrastructures, the usage of available grid computing power is tightly connected with particular research projects. Projects have to request processing power, requiring proposals to get hands on the requested resources. Another major difference concerns the resource management and programming model. Grids are dedicated to batch processing jobs. A user might, for example, submit a job which runs for 60 minutes on 100 processors. The scheduler then waits until 100 processors are available and executes the job for the given time. In a cloud computing environment, all applications/users operate at the same time, also enabling the execution of latency sensitive applications with an appropriate Quality of Service (QoS)-aware scheduler in place.

Keeping these key aspects in mind and prepared with the understanding of various cloud computing reference architectures, from different viewpoints and highlighting the key aspects [NIST, 2011], the NIST developed a more generic reference architecture [Lui et al., 2011], illustrated in Figure 2.1. This work will not reproduce the whole specification but will briefly introduce the involved actors and then focus on the orchestration model, as the most relevant topic for the further work on this thesis. The reference architecture defines five actors involved in the provisioning and operation of cloud services Cloud Consumer, Cloud Provider, Cloud Carrier, Cloud Auditor and Cloud Broker with their definitions in Table 2.1. These definitions again highlight the service-oriented nature of the overall cloud computing model. The actors work in close collaboration to provide consumers with their demanded services and also monitor their operational parameters, as well as performance and security of the implementation. More important in the context of this thesis is the service orchestration, which serves the cloud providers

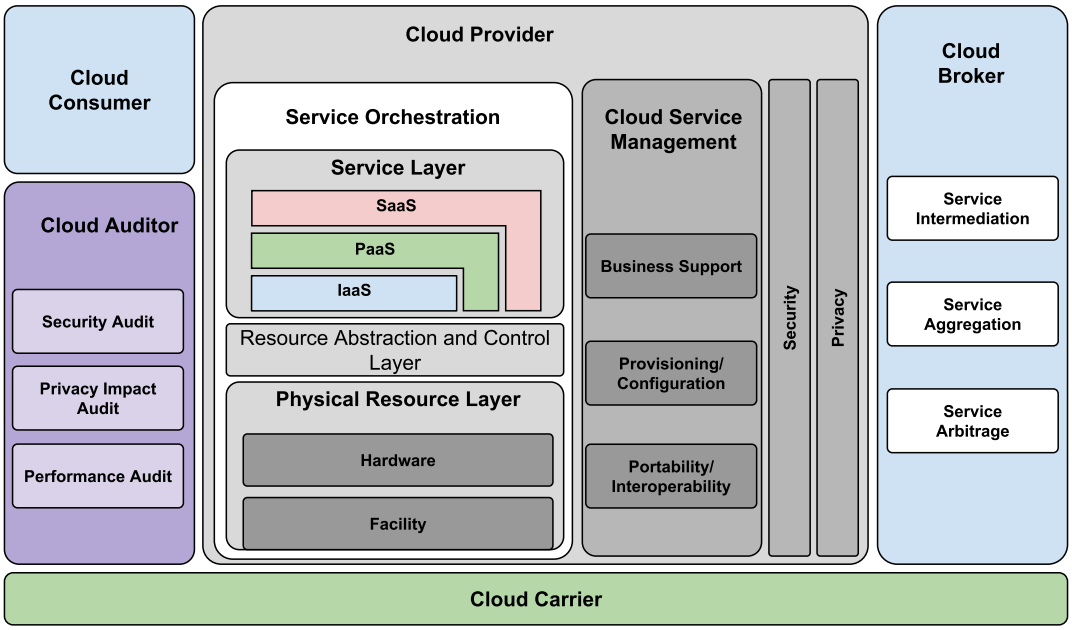


Figure 2.1: NIST cloud computing reference architecture [Lui et al., 2011]

to arrange and coordinate their resources to, finally, serve consumers their services. Orchestration follows a three-layered model for the final composition.

The *Service Layer* defines interfaces for the consumers. Three major service models have evolved:

- Infrastructure-as-a-Service (IaaS) provides fundamental functions like processing power, storage space, or networks capabilities. Consumers can use these provided resources to run arbitrary software. The prime example of IaaS is the acquisition of virtual machines for arbitrary tasks.
- Platform-as-a-Service (PaaS) provides a higher abstraction level than IaaS and enables to deploy applications, created using cloud provider supported programming languages. In contrast to IaaS, the consumer has no control over the underlying infrastructure. The prime example for PaaS is the deployment of web-based services by just uploading the application package, without exclusively setting up the operation environment.
- Software-as-a-Service (SaaS) goes one step beyond PaaS and only allows the consumer to use provided applications. The prime examples of SaaS are web-based applications like e-mail clients or online word processors.

The realisation of the defined service models can be dependent but also independent of each other. The PaaS model implementation could be built on top of IaaS and SaaS could be built on top of PaaS. Still, standalone realisations are possible. The impact of choosing one service model over the other for the consumer and provider are differences in their responsibilities. In a SaaS scenario, the cloud provider has full responsibility of deploying, configuring, updating and maintaining operation including automatic scaling according to the demands of consumers. For IaaS in contrast, the cloud provider acquires the (physical) resources like processing power, storage and network. The operation, updating and scaling of deployed services is up to the consumer. This process may again involve requesting new IaaS resources from the cloud provider, but responsibilities are shifted to the consumer. In a PaaS scenario, the cloud provider is responsible for a middleware layer like software execution stack, or databases. Still, the consumer has control over certain operation environmental properties but usually no control of the infrastructure level.

The *Resource Abstraction and Control Layer* provides an abstraction of the physical layer to the upper layers. It manages access to physical resources by using software abstractions of physical hardware.

Table 2.1: NIST Cloud Computing Reference Architecture actor definition [Lui et al., 2011]

Actor	Definition
Cloud Consumer	A person or organisation that maintains a business relationship with, and uses service from, *Cloud Providers*.
Cloud Provider	A person, organisation, or entity responsible for making a service available to interested parties.
Cloud Auditor	A party that can conduct independent assessments of cloud services, information system operations, performance and security of the cloud implementation.
Cloud Broker	An entity that manages the use, performance and delivery of cloud services, and negotiates relationships between *Cloud Providers* and *Cloud Consumers*.
Cloud Carrier	An intermediary that provides connectivity and transport of cloud services from *Cloud Providers* to *Cloud Consumers*.

This involves the usage of software components like hypervisors, virtual machines, virtual storage, and other virtualisation concepts. As illustrated by the mentioned components, heavy usage of virtualisation is a common implementation strategy for this component. Still, the model does not restrict the usage of other strategies. Usage control, metering, and access control is provided by the control aspect of this layer.

The lowest layer, the *Physical Resource Layer*, contains the physical, non-virtualised, CPU, storage, and network components but also facility components required for their operation like air conditioning, power supply and external communication facilities. The nested representation in Figure 2.1 does not only show the high-level composition of single layers but also refers to layer interdependencies by their stacked representation.

With this targeted introduction to cloud computing, a clear connection to the related grid computing concept was drawn. Both concepts follow similar goals. Grid computing tries to reach the vision by sharing available resources with others in a power grid like structure and strongly adhering to the resource sharing, which might be one reason why grid computing never made it into the commercial market. Cloud computing, in contrast, shifts its focus from providing resources, to providing services, which automatically evolved to great business opportunities. Today, cloud computing is already present for several years, still successful and evolving but facing new challenges especially from the mobile sector.

2.3 Mobile Edge Computing

Researchers already noticed the upcoming new challenges regarding mobile users. Mobile device users demand more processing power at the tip of their hands also usable for interactive operations. The crux about interactivity is not providing massive loads of processing power, as it is the case in grid- and cloud computing, but also reducing the latency from the mobile devices to the computing units to a minimum. Edge- or fog computing, in general, is a new concept targeting these issues. It adds another resource layer to the cloud computing concept in the proximity of the users, therefore having low latencies. Due to the proximity to the user, fog computing also enables the consideration of user relevant context information. Luan et al. [2016] illustrate fog computing as shown in Figure 2.2. Edge computing is not meant to replace cloud computing but can be considered as a mobile device focused extension. It differentiates from cloud computing as illustrated in Table 2.2. The fog computing concept

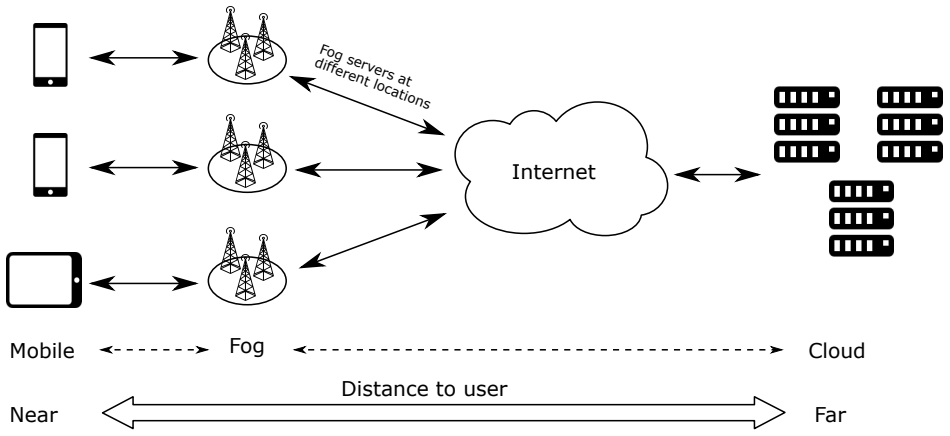


Figure 2.2: Fog computing overview [Luan et al., 2016]

Table 2.2: Key differences of cloud- and edge computing [Luan et al., 2016]

	Fog Computing	**Cloud Computing**
Target User	Mobile users	General Internet users
Service Type	Limited localised information services related to specific deployment locations	Global information collected from worldwide
Hardware	Limited storage, compute power and wireless interface	Ample and scalable storage space and compute power
Distance to Users	In the physical proximity and communicate through single-hop wireless connection	Faraway from users and communicate through IP networks
Working Environment	Outdoor or indoor	Warehouse-size building with air conditioning systems
Deployment	Centralised or distributed in regional areas by local business (local telecommunication vendor, shopping mall retailer, etc.)	Centralised and maintained by Amazon, Google, Microsoft, etc.

is meant to provide the initial idea without really diving into details of the implementation. The concept was picked up by the European Telecommunications Standards Institute (ETSI) and transformed into a technical specification [ETSI, 2016a; ETSI, 2016b; ETSI, 2017]. The general fog computing concept foresees a communication with the fog provided resources using local technologies like Wifi or Bluetooth but also using 4G and 5G networks. Mobile Edge Computing (MEC) in contrast, is completely tied to telecommunication providers and their deployed 4G and 5G communication technologies. On the one hand, this is a narrowing approach, on the other hand, it enables a detailed and technical specification of the capabilities, use cases, and used techniques.

An overview of the architecture defined by ETSI [ETSI, 2016b; Hu et al., 2015] is shown in Figure 2.3 and illustrated in the following. From an abstract point of view, applications installed on users' devices (user application) can request the instantiation of MEC applications on the MEC infrastructure. The user application is then able to use the MEC application-provided functionality. In their reference architecture, the authors differentiate between system- and host level. The system level acts as the contact point for user applications, whereas the host level represents the backbone and combines components

regarding the organisation of available computing resources. The authors of the MEC architecture define several logical entities:

- The *Mobile Edge Host* provides the storage, compute, and network resources for mobile edge applications by relying on virtualisation technology and hosts the mobile edge platform.
- The *Mobile Edge Platform* represents the hosting environment for mobile edge applications to offer their services, as well as discover and consume other services.
- *Mobile Edge Applications* run on the virtualised environment using virtual machine technology provided by the mobile edge host, ready to be used by user applications.
- The management components acquire virtualised hardware and instantiate the virtual machines. Furthermore, they manage the lifecycle of mobile edge applications and inform the system level components of important occurred events.

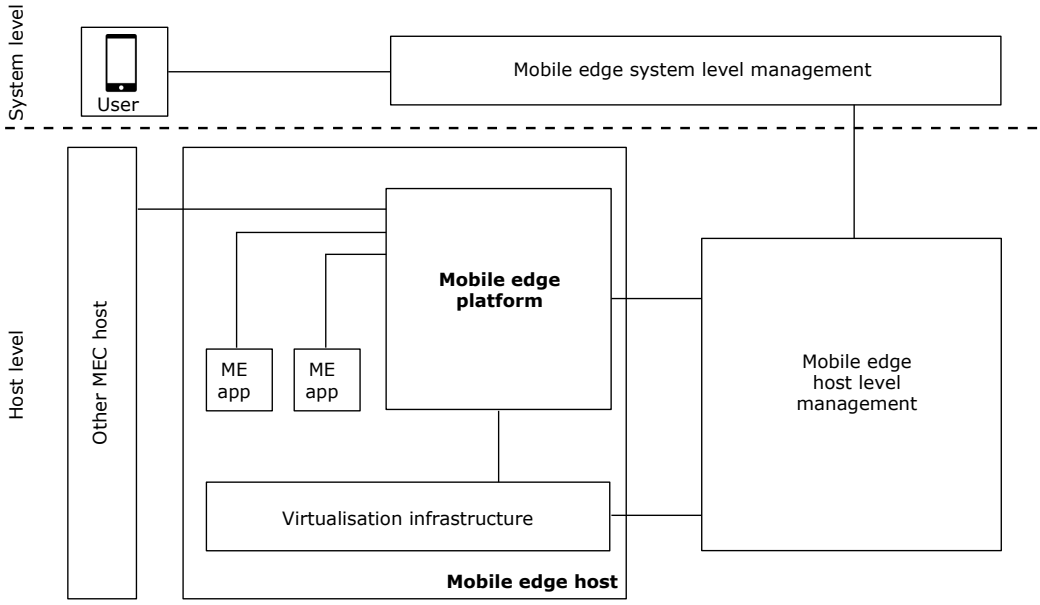


Figure 2.3: High-level MEC architecture

The anatomy of use cases for MEC to be realised as mobile edge application [ETSI, 2015] have the requirements for cloud computing supported backends but with low latencies from the user application to the computing units. Furthermore, applications gain additional possibilities by having access to contextual information for the user via the 5G access nodes. Prime use cases include the following:

- *Connected Cars:* The communication between cars on the road for the purpose of transmitting hazard warnings can be improved by using mobile edge applications. In this case MEC offers its proximity capabilities and analyses data received from cars, as well as roadside sensors and warns passengers if incidents are detected.
- *Video Analysis:* MEC can offer video analysis applications for surveillance cameras. This prevents slow and inaccurate analysis at the source and bandwidth demanding uploads to cloud computing centres.
- *Augmented Reality:* With the augmented reality technique, points of interest can be enhanced with useful information on a user's mobile device. This is a highly localised and latency sensitive process and requires exact positioning. Therefore, is best suited to be mapped to MEC enabled applications.

This brief introduction clearly shows the power of involving proximate computing resources and the positive effect of decentralisation concerning possible applications, as well as enhanced user experience.

2.4 Chapter Conclusions

In this chapter, a clear path from the first large-scale distributed systems, following the grid computing paradigm, to today's cloud computing and MEC environments is shown. The baseline technology for today's systems still has its ancestors in the grid computing domain. Nevertheless, grid computing never achieved broad commercial success, due to the absence of a suitable business model. By relieving the focus on sharing low-level resource and instead concentrating on services at different abstraction levels, cloud computing is still a large commercial success.

Recently it was discovered that the semi-centralised structured cloud computing is not capable of optimally serving the ever increasing demands of mobile device users, mainly due to high latencies. MEC targets this issue by providing architectures and frameworks for a decentralised cloud-like environment in the proximity of the users' devices. This novel paradigm is tightly connected to telecommunication providers and their already existing dense Radio Access Network (RAN). MEC is capable of adding great value to applications concerning contextual information, user experience due to reduced latency, and reliability. Still, from a computing point-of-view the differences from MEC to cloud computing are limited. As it is the case in cloud computing, MEC requires to statically and explicitly define interfaces between parts executed using MEC and applications executed on users' devices. Basically, MEC shifts the semi-central deployment of cloud services close to the user.

Chapter 3

The Evolution of Hybrid Mobile Edge Computing

In the previous Chapter 2 the evolution of today's prevalent cloud computing technique is illustrated, and a changing environment towards mobile devices is anticipated and quickly approaching. The technical realisation of MEC is a novel contribution in the sector of cloud- and distributed computing but the importance of low latencies and the concept of using proximate computing power was already discovered several years ago. A major issue of MEC manifests in a massive overhead for software developers to make use of this new technology. Mobile edge applications need to be explicitly separated from applications running on the mobile devices. This results in a static and enforced application partitioning into locally (on the mobile device) and remotely (as a mobile edge application) executed parts. More flexible execution models are required to optimally adapt to the rising mobile device usage paradigms.

Research in this direction has already been performed long before the concept of MEC was developed, is still active and is more relevant than ever. In this chapter, approaches to enhance the capabilities of mobile devices with resources, like storage or computing resources, are discussed. Furthermore, their connection and future importance to MEC is highlighted.

3.1 Cyber Foraging

Cyber foraging represents a concept to dynamically enhance resource constrained devices with computing resources from other, more powerful devices. Satyanarayanan [2001] first described this concept as a solution to some of the challenges of the pervasive computing paradigm. Pervasive or ubiquitous computing targets at the fusion of human users and their omnipresent mobile devices. Users receive assistance and useful information from their devices, but the devices remain invisible and do not push into the foreground. The pervasive computing paradigm depends on other fields of research but also opens new research areas. A visualisation and identification of the relevant research areas from Satyanarayanan [2001] and their relations are shown in Figure 3.1. Pervasive computing, in particular, opens the following high-level research areas to the community:

- *Smart spaces:* Smart spaces are enclosed areas like meetings rooms. The effective usage of smart spaces includes automatic adjustment of light according to the users' requirements or different behaviour of software depending on where the user is currently located.
- *Invisibility:* In an ideal pervasive computing setting, devices completely disappear from the users' consciousness and remain invisible.
- *Localised scalability:* With a growing acceptance and sophistication of smart spaces, the number of interactions increases and higher bandwidths are demanded. This issue is increased with the

presence of multiple users. Therefore, scalability in the users' surrounding is key for pervasive computing.

- *Uneven conditioning:* The introduction of novel technologies takes time, especially in the beginning, to reach a critical mass of early adopters. The rate of penetration, as well as the quality of the equipped smart spaces, will greatly vary. Maintaining a user's pervasive environment requires ways and techniques to mitigate the effects of unavailable smart spaces.

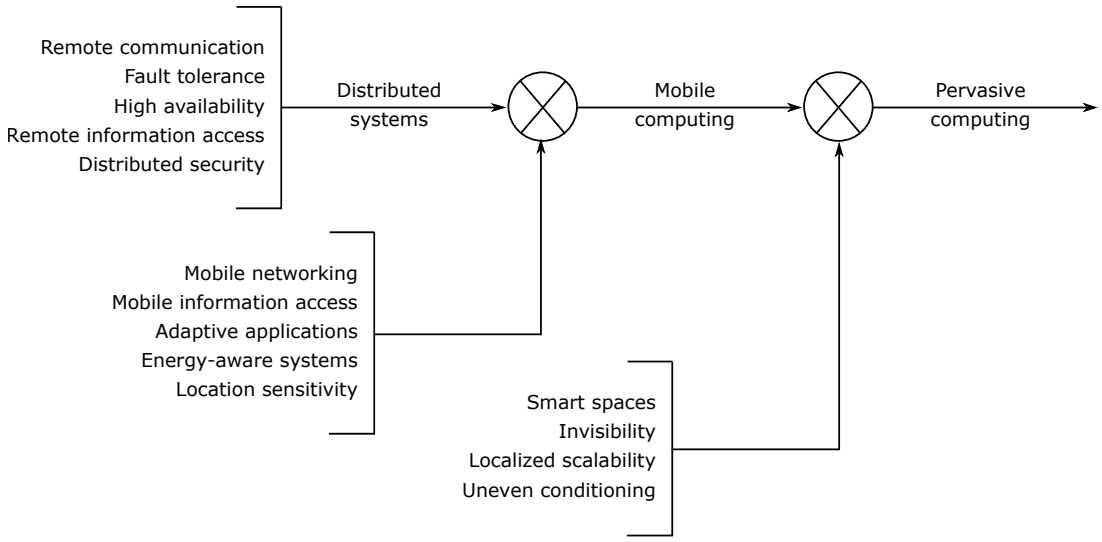


Figure 3.1: Research areas in the pervasive computing domain [Satyanarayanan, 2001]

Furthermore, Satyanarayanan [2001] developed two scenarios which show the potentials of pervasive computing. The technological building blocks were already available at that time. Still, the scenario sounded like a story from a future world. The scenarios introduce Aura, a pervasive computing system, with the ability to act pro-actively by correctly identifying a user's intent, self-tuning by adjusting to the user's environment and involving smart spaces and other equipment to seamlessly move applications between devices and computers. The surprising result is that all the required technologies are already there. They need to be combined to overcome the main obstacle, a massive data analytical challenge. Another result is the identification of requirements on the used mobile devices: they need to get smaller, lighter, and require an extended battery lifetime to utilise the full potentials of the pervasive computing paradigm.

Developing smaller and lighter mobile devices has the obvious effect of reduced computing resources for an equal technological level. Cyber foraging is a concept to counter these losses by dynamically augmenting a mobile device's computing resources with other available computing resources. This will include an exploitation of smart spaces at different sites. The authors envision a typical scenario as follows:

- Once entering a new neighbourhood, potential candidates are identified using a suitable communication technology.
- The application area is negotiated with the identified computing resources.
- Does a particular computing resource only act as a cache or high-bandwidth gateway to the Internet or does it even allow to execute arbitrary application code in application offloading scenarios?

This approach raises many questions and requires extensive research in different areas. The following is only an indicative listing of some major fields of research: How does the discovery process work? How

to establish trust with the identified computing resources? How much time is required by computing resources to provide an application-compatible execution environment and how can it be improved? How to scale the smart spaces approach?

The authors do not provide any solutions to the identified issues at that time but raise critical questions and motivate a whole new research area. The overall concepts are not a novel contribution: distributed-, mobile-, and pervasive computing were known before. They are brought into the emerging smartphone sector. The concept was picked up by Goyal and Carter [2004] and Lewis et al. [2014]. They provide solutions for a fraction of the raised challenges and issues, also mixing the concept with recent innovations like Cloud Computing (CC). The security issues are only scratched on the surface and limit to end-to-end security with already established trust relationships. From today's point-of-view most parts of the original vision are already realised but using different techniques. Today, scalability is reached by exploiting CC, with new techniques like MEC in the pipeline. The original vision of smart spaces is not realised yet, but the resulting functionality can still be provided. Google and Apple are the major players on the smartphone market with high penetration rates. Recording the habits, data and location of users, combined with machine learning approaches, allows them to accurately predict users' intentions and provide useful context information like:

- Provide automatic weather forecasts.
- Estimated time for a user to arrive at its destination, without explicitly entering its destination.
- Automatically remind users about upcoming events like "leave now to catch your flight" or "start now to be on time at your appointment" even considering the preferred transportation.

The visions are actually not reached using the cyber foraging paradigm but by using the semi-central CC approach, which is further elaborated in Chapter 3.2. The vision of cyber foraging, including its smart spaces and local scalability property, is still up-to-date. A possible approach which again picks up this paradigm is MEC from Chapter 2.3.

3.2 Mobile Cloud Computing (MCC)

MCC acts as an enabling technology for the pervasive computing vision as provided by Satyanarayanan [2001]. In the sector of MCC and CC in general, terms are often mixed up. Therefore, a clear definition is required. Particularly MCC is often mistaken for cyber foraging and other related technologies. In this thesis, MCC is defined following the illustration in Figure 3.2 and according to Fernando, Loke, and Rahayu [2012]. Remote resource-rich servers run an application, where mobile devices act as thin clients over the available Internet connection. The realisations can have different shapes and represents a trade-off between available computing resources and flexibility concerning the availability of an application if the cloud is not in reach.

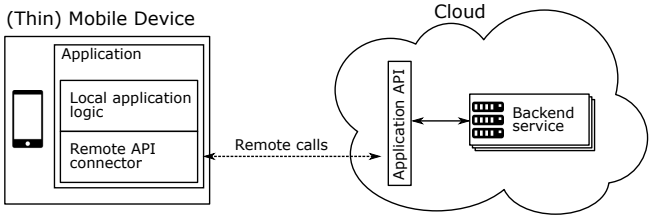


Figure 3.2: Mobile Cloud Computing overview

The commodity of all realisations is the static partitioning of applications regarding the locally and remotely executed parts. Interfaces between local and remote parts are statically defined in the development process. Considering a mobile voice recognition application, where voice samples are uploaded to

a remote service deployed in the cloud for analysis and the analysis results are sent back. In this case, the mobile device just records the samples without any analysis logic. Voice recognition is a computationally intensive operation. Offloading this operation to external computing units is a clever strategy to save energy on the mobile device and speed up the operation. The downside is a dependency on the cloud service. Without the service, the application is not operating at all.

The MCC approach has a clear connection to CC in general, but CC is much more than MCC. It is the specific use case what makes the difference from CC to MCC. Backend services (primarily) for mobile applications with scalability features can be considered as MCC from a mobile device's perspective. Thereby, it is irrelevant how services are deployed: IaaS, PaaS, or SaaS. From a pervasive computing perspective, MCC can be seen as an intermediate step between the cyber foraging vision and the realisation using the HMEC approach.

To draw a complete picture of the MCC technique, a second approach needs to be introduced. Although this thesis refers to the above definition of MCC, existing frameworks occasionally link to MCC as a more advanced service enabling a dynamic offloading of application logic. Applications are still partitioned into locally and remotely executed application parts but, in contrast to the approach described above, the whole application logic is also included in the mobile application. This enables to dynamically decide if particular application parts should be executed locally or remotely under current mobile device conditions. The dynamic decision enables the operation of MCC applications in scenarios where no cloud connection is available but also raises major questions regarding security.

Due to the connection of CC and MCC, the security issues are also inherited. Security issues of MCC are strongly related to security issues in the HMEC domain, therefore are scrutinised throughout this thesis.

3.3 Hybrid Mobile Edge Computing (HMEC)

HMEC is introduced by Reiter, Prünster, and Zefferer [2017] with a strong connection to MCC and Hybrid Mobile Cloud Computing (HMCC) [Abolfazli, Gani, and Chen, 2015]. HMCC can be considered as an intermediate step between MCC and the proposed HMEC model, also considering a device's surrounding up to a limited extent and still heavily relying on CC infrastructures. HMEC realises the original vision of cyber foraging concerning the computing resource usage by combining the concepts of CC, MEC, MCC and mixing in some ideas from HMCC. Today, telecommunication providers are omnipresent which is proposed to be exploited by MEC. MEC envisions to deploy computational resources at the telecommunication base stations to be used by mobile edge-enabled applications. Applications executed on mobile devices are prepared to work with mobile edge application counterparts, deployed on the MEC infrastructure. In fact, this concept works analogue to MCC with mobile edge applications dynamically deployed close to the mobile device. HMEC, in contrast, refrains from defining new pro-

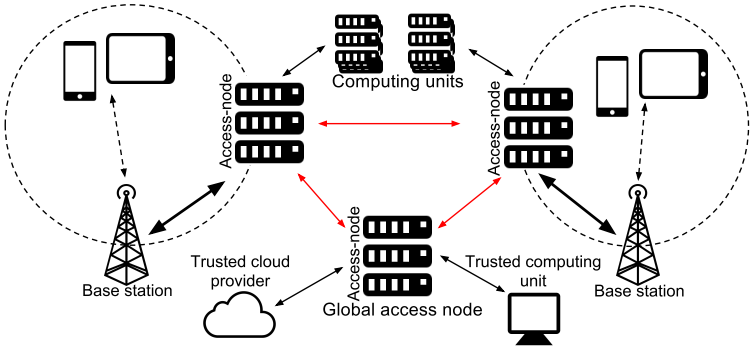


Figure 3.3: Hybrid Mobile Edge Computing overview [Reiter, Prünster, and Zefferer, 2017]

gramming models. Still, it targets at a broad usability of available computing resources on mobile devices and does not exclusively tie the resource usage to telecommunication providers. A HMEC overview is provided in Figure 3.3. If supported, telecommunication providers' base stations could play a major role in the discovery process of computational resources but HMEC does not depend on them. The concrete functionality and inner workings of this concept is developed in this thesis, acting as one of the main contributions. Overall, the concept enables to realise the introduced smart spaces vision and beyond. Locally available computing resources can be used by mobile devices to enhance their computing power and save energy. In case no free local computing capacity is available, the system enables to fall-back to global CC units or to a local only operation. To have a clear view about the available computing resources, a clever organisation of computing units is required. This multi-levelled approach requires flexible applications, capable of dynamically switching their execution or to an available computing unit.

Flexibility also comes at a certain price. Approaches following the previous MCC model only have a single computing resource where executions take place, representing a small and controllable attack surface. Executing on different computing units also poses a large attack surface. Frameworks need to assure that users' data and computations are kept safe.

3.4 Verifiable Computing

A different field of research, currently being highly active, is the field of verifiable computing. It does not target (like MCC and HMCC) the distribution of tasks to other computational resources but focuses on how to ensure that a client device receives the correct result from an offloaded computation. Yu, Yan, and Vasilakos [2017] provide a survey and current state of the research in the field of verifiable computing. The problem with current schemes in this field of research can be summarised as the following:

- Schemes are only efficient for a limited set of function in certain classes.
- Some schemes rely on complex cryptographic primitives like fully homomorphic encryption schemes and have high computation efforts. [Catalano and Fiore, 2013; Yan, Yu, and Ding, 2017] Efficiency is key when operating on resource-constrained mobile devices.
- Another key aspect about verifiable computing schemes is their applicability to already existing applications. Considering existing MCC and HMCC frameworks as illustrated in the following chapters, minimal or no effort is required to enable offloading of already existing applications. Verifiable computing approaches require a particular representation of functions considered for offloading. The approach introduced by Gennaro, Gentry, and Parno [2010] requires functions to be available as boolean circuits, an additional pre-processing step. Pinochhio [Parno et al., 2013], to name another example, is closer to being a practical solution in terms of performance, but requires quadratic programs [Gennaro, Gentry, Parno, and Raykova, 2013] for execution.

This thesis focuses on applicability and achieving security goals using infrastructure, architectural and technical means. Still, verifiable computing is considered as an interesting and promising field of research for future developments.

Chapter 4

Part I Conclusions

Currently, two relatively disjunct research areas can be observed. On the one hand distributed computation models like grid computing were evolved to a service and customer oriented CC paradigm, on the other hand researchers focussed on developing mobile application execution models to enable a flexible and dynamic execution in a mobile centric world. The goal of these execution models is to keep up with users' demands. The utilisation of mobile device has massively increased, also for business related tasks, affecting the battery lifetime of mobile devices. The defined execution models aim at offloading computationally intensive tasks to resource rich computing units with an energy-saving effect but also positive performance impact on the mobile devices.

MEC is the first approach so far trying to close the gap between these research areas. The MEC solutions makes use of the already existing infrastructure of telecommunication providers but make tradeoffs at the mobile application execution models:

- The MEC model is tied to the telecommunication company-provided infrastructures. This does not represent a valid realisation of smart spaces.
- The flexibility of MEC is hindered by not integrating with other available computing resources. Although MEC, as defined by ETSI, supports seamless mobile device handovers from one telecommunication domain to another, other resource types like classic CC or custom user specific resources are not utilised.
- Mobile device applications following the MEC model are always composed of a user application executed on the mobile device, and mobile edge applications executed on the MEC infrastructure. This approach hinders a dynamic decision regarding a local or remote execution of particular application parts, based on a device's environment.

With the introduction of HMEC these weaknesses are sorted out. HMEC deployments could make use of available telecommunication infrastructures but do not depend on their availability. Furthermore, it enables the usage of different resource types like CC-provided resources, custom user resources, or proximate resources. MEC contributes to the HMEC model as an invaluable resource provider. Without considering the technical realisation and possible technical limitations at this stage, HMEC serves as an enabling technology to realise the vision of smart spaces.

The gained flexibility comes at a price. Substantial obstacles need to be overcome, summarised at a high abstraction level in the following listing:

- *Trust and Security:* The existing, relatively static mobile application execution models only considered trust and security as a side note, limiting their contribution to employing end-to-end security techniques with pre-established trust relationships. Using a more flexible and dynamic approach

also increases the importance for state-of-the-art and dynamic security mechanisms. Users need confidence that their processed data is safe and not exposed to unauthorised parties.

- *General Architecture:* Grid computing and CC only have loose or multiple architecture definitions. For MEC, in contrast, ETSI defines a detailed architecture. HMEC, as a novel model, does not have a uniform architecture definition yet.
- *Interoperability:* Due to the wide spectrum of considered resource types, HMEC has to deal with different architectures and operating systems. For a seamless user experience HMEC needs to be available on all platforms, also cross-platforms.

These identified high level issues are scrutinised in the remainder of this thesis and concepts, models, architectures, and implementations are developed.

Part II

A Security Assessment Framework for (Hybrid) Mobile Cloud/Edge Computing Frameworks

Chapter 5

Existing (H)MCC Frameworks

The borders between the concepts of MCC and HMCC are floating. Only very few existing frameworks are considered as true *hybrid*. Still, this does not render MCC frameworks or frameworks following a different approach less essential for the development and improvement of HMCC approaches. Therefore, this chapter provides an overview of frameworks relevant for enhancing the HMCC approach. The relevancy of frameworks is determined by their scientific significance and their provided innovative factor. Nevertheless, to maintain a certain level of comparability, a similar technological level is required. In contrast to Chapter 3, where the underlying high-level models are introduced, this chapter focuses on concrete, representative frameworks. They are presented here to a level which enables (a) a security focused evaluation in Chapter 6, and (b) the identification of reusable technologies for the process of further improving HMCC. The following is a listing of frameworks introduced in this chapter, also providing motivations about the selection of each particular framework:

- *MAUI* [Cuervo et al., 2010] is the first concrete implementation targeted at a new device class, smart devices. It enables to dynamically offload computationally intensive parts of an application to remote cloud-supported computing units but requires changes on the source code level. It acts as the baseline for most later researches in this area.
- *CloneCloud* [Chun et al., 2011] also enables offloading of computationally intensive application parts but operates on unchanged applications without requiring access to the source code.
- *Cuckoo* [Kemp et al., 2012] enables to integrate multiple implementations for the same task, targeted at different architectures and environments. This way the targeted computing unit can be better utilised, in case an optimised implementation is available.
- *ThinkAir* [Kosta et al., 2012] builds on MAUI and CloneCloud and adds scalability by enabling an on-demand start-up and shut-down of virtual machines in the cloud, as the application demands more or fewer resources.
- *TANGO* [Gordon et al., 2015] identifies weaknesses in the partitioning process, resulting in particular characteristics of offloaded applications like a high computational load for tasks. The authors propose a new approach, not relying on a static partitioning, but dynamically switching between multiple executed replicas.
- *mCloud* [Zhou et al., 2015] shifts the anticipated unrealistic operation environment of previous frameworks to more realistic scenarios by also considering nearby devices and not statically binding to single cloud-supported computing units.

This chapter continues on the analysis carried out by Reiter and Zefferer [2015a]. The analysis of the frameworks in [Reiter and Zefferer, 2015a] is extended and enhanced with required details. Frameworks,

which are not considered relevant in the context of this thesis are omitted. Furthermore, new frameworks are analysed, following the same approach.

5.1 MAUI

MAUI has been introduced by Cuervo et al. [2010] with the primary goal of offloading computations to save energy. It uses a method-level offloading approach and hence provides fine-grained outsourcing opportunities. Methods that can be offloaded are marked as *remotable* by applying source-code level annotations. These annotations are extracted at runtime and the execution is dynamically intercepted. An optimisation framework is utilised to decide if a method should be offloaded under current conditions such as available connectivity, Round-trip-time (RTT), or effort required to transfer the currently required state of the application. To enable these transparent interceptions, MAUI is limited to managed code environments, specifically to the Microsoft .NET environment.

MAUI's high-level architecture is illustrated in Figure 5.1. It is separated into client and server components, but with very similar behaviour. The executed application, which is also available on the server, is attached to the MAUI runtime module. The runtime module performs the required steps to delegate the execution to a remote server or performs the execution locally. The following components are involved:

- The first component, which is not part of the runtime module, is the annotation system. The annotation system enables the application developer to perform a suggested partitioning of the application into parts considered for offloaded execution and parts where the execution needs to be performed locally.
- The *Profiler* component works in close collaboration with the *Solver* component. The *Profiler* component collects profiling data on (a) device level: the estimated energy consumption for a given method, (b) program level: transfer requirements for each method, runtime duration, CPU cycles, and (c) network level: RTT, bandwidth, packet-loss. This profiling data is fed into the *Solver* component.
- The *Solver* component is, due to its complexity, split between the server and the client. The actual decision is made at the server. Based on the collected data, a model was developed to decide if a remote execution is beneficial.
- The *Proxy* components perform the actual offloading process. They are generated during compilation of the application, considering the remotable methods. Required data is transferred to the server, instantiated, executed, and the result is transferred back.
- The *MAUI Controller* supervises all actions and performs other relevant activities like authentication.

Design decisions were made for the MAUI prototype to support offloading to different architectures. Smartphones, for example, operate on ARM CPUs, desktop computers, in contrast, mainly use Intel x86 compatible CPUs. To make the offloading operations interoperable, MAUI limits to managed code environments and does not allow offloading of native calls. If the connection to the MAUI server is lost during an offloading operation, the framework restarts the process locally, resulting in a short delay only.

MAUI mainly focuses on the offloading part and ignores details on how to manage or combine available resources. The smartphone has a static connection to the MAUI server. The most interesting aspect of MAUI is its simplicity. No prior knowledge is required to make use of this framework. Only few internals need to be known by the developer to improve the execution of applications using MAUI. Although MAUI focuses on the .NET programming language, it can easily be adapted to other programming languages. Unfortunately, the .NET framework is still not very popular on mobile devices and basically

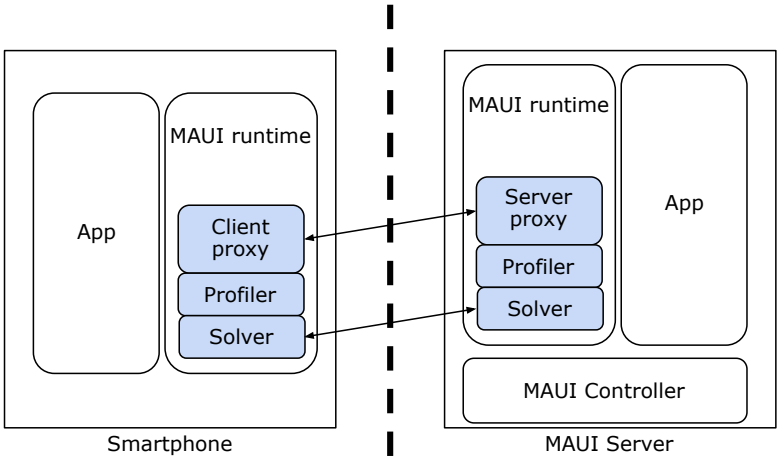


Figure 5.1: MAUI highlevel architecture [Cuervo et al., 2010]

restricted to the Windows Phone platform. Although there are ways to execute .NET applications also on other platforms, it is not very popular at the moment and may not be of interest for platform vendors. Mobile devices do not only differ regarding their architecture but also concerning their operating systems. Mobile device operation systems developed in the direction to support only a very limited set of programming languages. These programming languages are often attached to a whole ecosystem and community, e.g. Android applications are best developed using the Java programming language. MAUI is operating on the source code level, therefore porting to different programming languages, supported by mobile operating systems, is required to achieve a broad compatibility. This results in different silo solutions requiring a compatible execution environment. Therefore, they are not compatible with each other.

Regarding security, MAUI relies on a pre-established trust relationship. This perfectly fits into the goal of MAUI to focus on the offloading technology but leaves a large gap in terms of security.

5.2 CloneCloud

CloneCloud is introduced by Chun et al. [2011] with the goal of offloading tasks to platform clones in the cloud. A minimisation of the energy consumption on the mobile device and an optimisation of the execution time is envisaged. The primary motivation for the authors to focus on this topic was the situation in the cloud computing sector at this time, and is still valid today. Bringing a rich application to mobile devices is a demanding task and tends to be inflexible: either an application is written as a huge monolithic blob and performs all the tasks on the mobile device, or is statically split using a traditional client-server approach moving the computationally intensive parts to the cloud. The chosen separation might be optimal for one mobile device and environmental condition set, but not optimal for another one.

These issues are addressed by CloneCloud, enabling an operation on unmodified Android applications. The CloneCloud system modifies existing applications and allows a dynamic migration of whole application threads to remote systems. The remaining application logic keeps executing on the mobile device while an application thread is offloaded, but blocks once access to state of the offloaded thread is required. If the thread completed execution on the remote host, the generated state is merged back into the mobile application and can be accessed by other threads.

In the case of CloneCloud, the partitioning of applications is an offline process, performed prior to the application deployment. CloneCloud features a static analyser and a dynamic profiler. Based on a set of execution conditions (network characteristic, CPU speed, and energy consumption) the static analyser

yields partitions to optimise the application for execution time or energy consumption. This process is performed for several different condition parameters, resulting in a partition database. Furthermore, the dynamic profiler collects data, used to create a cost model. The data is generated by invoking the profiler in multiple executions of the application, using randomly generated input data.

At runtime one of the recorded partitions is picked, according to anticipated condition parameters. The partition is implemented in the application binary by inserting special VM instructions, acting as migration and re-integration instructions. A high-level architecture of the CloneCloud system is illustrated in Figure 5.2: The *Migrator* component on the smartphone catches the inserted migration instructions with the goal of moving the execution to a remote server. Therefore, the *Migrator* collects the whole state of the target thread and passes it on to the *Node Manager* for data transfer. The *Node Manager* has a static set of potential remote cloud computing units, matches this set with the available partitions and transfers the state to the selected remote system. At the remote system, the data is passed on to the *Migrator* of a newly allocated process, and the capture process is reversed to resume the thread. The *Migrator* on the remote side catches the re-integration instruction and performs the identical migration process back to the smartphone.

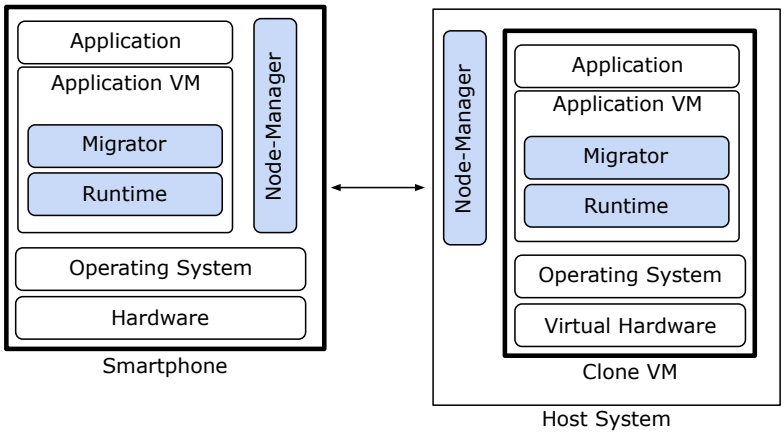


Figure 5.2: CloneCloud highlevel architecture [Chun et al., 2011]

The prototype introduced by Chun et al. [2011] is based on Java and Android. Therefore, it is theoretically able to operate on every available application. The migration and reintegration points are induced using special Java-VM commands. Therefore, for the framework to run, it is necessary to provide a modified Android version. No modified executable will run on unmodified versions of Android. Table 5.1 lists the current distribution of deployed Android versions, according to Google [2017]. As the distributions of currently used versions of Android is heavily scattered from version 2.3 to 7.1, it is unlikely that this technology will be used in practice. It requires a modified Android version on each deployed device.

From a security standpoint, CloneCloud is still in its infancy. CloneCloud focuses on improving the technical offloading foundation. Although the approach is capable of using multiple computing units, the node manager at the remote system acts as a trust anchor for mobile devices. A trust relationship with the node manager needs to be established prior to the application enrolment.

5.3 Cuckoo

Kemp et al. [2012] propose Cuckoo, an offloading framework for smartphones. Cuckoo does not aim at providing seamless operation for existing applications, but makes use of the natural partitioning of Android applications. Android applications are composed of the following fundamental components:

Table 5.1: Distribution of Android platform versions [Google, 2017]

Version	Codename	API	Distribution
2.3.3 - 2.3.7	Gingerbread	10	1.0%
4.0.3 - 4.0.4	Ice Cream Sandwich	15	1.0%
4.1.x	Jelly Bean	16	4.0%
4.2.x		17	5.7%
4.3		18	1.6%
4.4	KitKat	19	21.9%
5.0	Lollipop	21	9.8%
5.1		22	23.1%
6.0	Marshmallow	23	30.7%
7.0	Nougat	24	0.9%
7.1		25	0.3%

- *Activities* act as entry points for interacting with the user, and only do basic computations with low CPU utilisation. By blocking an activity, the user interface is blocked and gives the user the impression of an unresponsive application.
- *Services* in contrast do not provide user interfaces and are meant to perform long running operations and heavy computations. Activities can establish a binding to services and communicate through Inter-process Communication (IPC) using pre defined interfaces.
- *Content Providers* manage shared data sets. If a content provider gives its permission, other applications can modify data. Content providers are not elaborated in detail here, because they do not have a direct relevance for the offloading framework.
- *Broadcast Receivers* enable to deliver out-of-band events to applications, not contained in the normal program flow. Again, they are not elaborated in detail here, because they do not have a direct relevance for the offloading framework.

This natural separation between computational intensive and user interface parts of an application, with already well-defined interfaces, is exploited by the Cuckoo framework. The goal of the Cuckoo framework is to provide local and remote execution capabilities, but enabling custom implementations for different remote computing units to respect their peculiarities. During the build process, the Cuckoo framework hooks into the service API and adds a decision logic, whether to execute a service call locally or remotely. Currently, a basic decision logic is used: remote execution is always preferred. The offloading process is controlled by a resource manager, where all available computing units need to be registered beforehand. The application package deployed on the smartphone contains the complete code for local execution and different remote execution targets. On execution of a service call, the Cuckoo framework and resource manager decide if offloading should be performed, and transports the application logic and program data to the remote computing unit for execution.

The Cuckoo framework uses an interesting approach, which can easily be deployed to many Android applications. Providing different implementations for different architectures and environments is an elegant solution to get the best out of each computing unit. Nevertheless, this potentially requires a lot implementation effort. Furthermore, this approach is completely tied to native Android applications. This approach cannot be adapted for different operating systems or different programming models. Additionally, security capabilities are again limited to pre-established trust relationship, where each single device needs to trust all available resource managers.

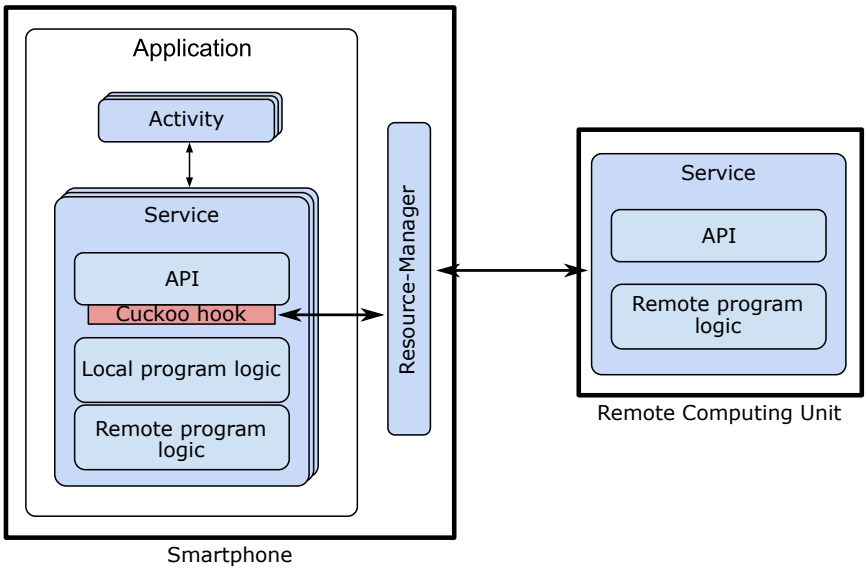


Figure 5.3: Cuckoo model [Kemp et al., 2012]

5.4 ThinkAir

Kosta et al. [2012] introduce the ThinkAir framework as an improvement of MAUI's and CloneCloud's concepts. Their main goals and novel contribution in the sector of application offloading is twofold: (a) They aim at providing an efficient way to perform on-demand resource allocation in the cloud, and (b) exploit parallelism by dynamically creating and destroying virtual machines. ThinkAir is designed based on four major principles:

- *Dynamic adoption to changing environments:* Offloading frameworks mainly target resource-constrained mobile devices, therefore frameworks need to dynamically adapt to changes in the environment.
- *Ease of use for developers:* In principal, the framework should be transparent to the application developers, without requiring developers to know the inner details of the offloading framework.
- *Performance improvement through cloud computing:* ThinkAir's main goal is to improve the performance and minimise the energy consumption of mobile applications on mobile devices.
- *Dynamic scaling of computational power:* The offloading framework needs to be able to react to changing computational requirements of mobile applications.

This results in an architecture as illustrated in Figure 5.4. ThinkAir provides a pre-partitioning mechanism, similar to the approach used by MAUI, and introduces an additional compilation step, which generates code for the remote execution. A proxy mechanism allows the *Execution Controller* to intercept method calls which could potentially be offloaded.

Once the *Execution Controller* intercepts a call, it decides based on the available profiling information if it is beneficial to offload a call:

- Historic execution times acquired by the *Program Profiler* are used to assess future execution times and get an accurate estimation.
- Data on consumed energy of past executions, collected by the *Device Profiler* is also considered in the decision. Offloading only takes place if a decrease in energy consumption is anticipated.

- Depending on offloading goals, decisions are based on weighting execution times and energy consumption.
- Cost of the consumed cloud resources can also be a factor considered in the decision.

If the final decision is to offload a call, Java reflection is used to transform the relevant application state to the application server.

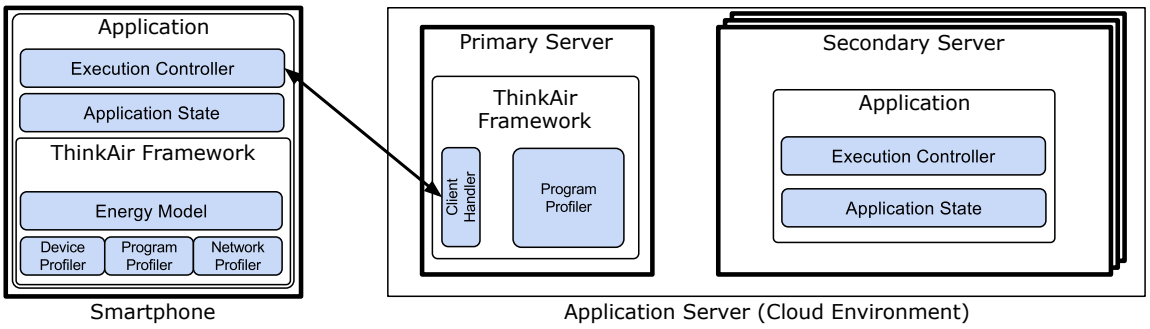


Figure 5.4: ThinkAir architecture overview [Kosta et al., 2012]

The *Application Server* can be considered as a cloud environment composed of multiple virtual machines. The *Primary Server* is always active, with a *Client Handler* waiting for incoming connections and accepting offloading requests. Once offloading requests are received, *Secondary Servers* are started on demand, which perform the work and return the result.

ThinkAir eliminates scalability limitations of previous approaches, still a major overhead due to the dynamic provisioning of virtual machines remains. From a security perspective, this approach is predestined for pre-establishing a trust relationship with the *Primary Server*. The *Primary Server* enjoys ultimate trust of the mobile device and is responsible to assess the trustworthiness of the *Secondary Servers*. This approach is reasonable in the ThinkAir scenario but does not serve more dynamic procedures.

5.5 TANGO

Gordon et al. [2015] identify deficiencies in the current model used by offloading frameworks. Due to their partitioning-based approach, offloading frameworks have the challenging task of accurately predicting the runtime of application parts. The success of offloading frameworks inherently depends on an accurate estimation. Therefore, Gordon et al. [2015] propose the TANGO framework, which follows an improved approach to increase the user-perceived mobile device performance.

TANGO proposes an offloading model based on replication instead of performing partitioning. Using replication, an application is simultaneously executed on the client and the server. At every point in time, one of the replicas is leading and takes over the part of performing computations and network communication, as well as displaying content to the user. Their approach to improve user-perceived performance is based on five main techniques:

- Each of the replicas is capable of sending output directly to the user. If, for example, a remote replica is in control, it can send the generated output to the user's screen, even though the local replica has not reached this state yet.
- Deterministic replay approaches are used to ensure that all replicas produce the same output. Non-deterministic events are logged by the currently leading replica, and are used for replay on other replicas.

- Using TANGO, the logging of non-deterministic events is spread among different computers. Especially when dealing with mobile applications, some input sources (e.g. user input or GPS data) are tied to the local device, whereas network input and output may be logged at a remote server.
- To reduce the required log operations of non-deterministic events, TANGO replicates I/O sources, data storage functionality, file system, and databases.
- In contrast to other replication-based systems, TANGO allows the leader role to float between replicas, also called flip-flop replication.

TANGO targets at unmodified interactive applications for the Android platform using the Dalvik VM. The ultimate goal of TANGO is the improvement of user-perceived performance. An indicator for the performance, is the required time from user input to the expected user interface feedback. For some sample applications, the authors achieved a latency reduction of more than 60%. The energy consumption in contrast nearly remains at the same level for most test cases.

The authors follow an interesting and new approach, still it suffers from the similar issues than partition-based systems like high connection latencies and limited bandwidth. Due to the fact that flip-flop replication requires to synchronise multiple replicas, the vital reduction of energy consumption on mobile devices seems out of reach. The authors put their focus on the novel offloading approach and do not consider security aspects.

5.6 mCloud

Zhou et al. [2015] introduce the mCloud framework, where the focus is not completely put on application partitioning and offloading technique. For partitioning, they use a similar source-level annotation approach as the previously introduced MAUI and ThinkAir framework. mCloud operates on Java for Android and dynamically captures annotated offloading candidates using Java reflection. These approaches were already elaborated in this chapter.

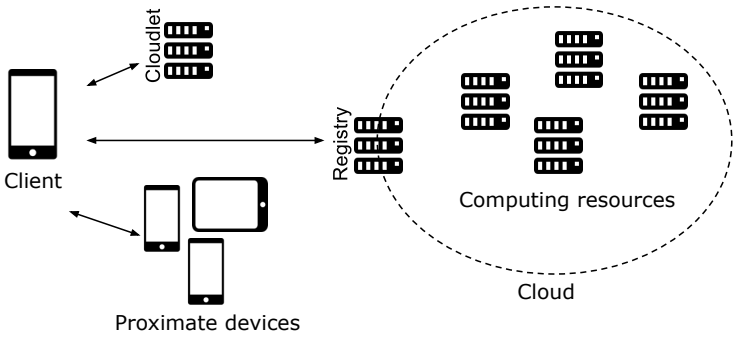


Figure 5.5: mCloud overview [Zhou et al., 2015]

As illustrated in Figure 5.5, the novelty of mCloud is the inclusion of cloud computing units and nearby available computing power. For cloud computing units, a central registry is setup where clients can acquire available computing units. To discover nearby resources, the client sets up an ad-hoc network, to enable a discovery of other mobile devices. Once other devices are detected, ad-hoc connections are formed and the client device can use the other devices for its offloading operations.

Although massive savings in energy consumption are achieved, this approach imposes multiple issues. As the authors discovered, nearby mobile devices change frequently and therefore the approach requires numerous renegotiations of the ad-hoc networks. Furthermore, standard smartphones are not capable of settings up an ad-hoc network and being connected to another wifi network. This results in the

situation that client smartphones are tied to their 3G or 4G connections with potentially higher latencies and lower throughput.

From a security perspective, mCloud requires sophisticated and dynamic approaches, matching its dynamic computing resource acquisition but security is not considered by the authors.

5.7 Chapter Conclusions

In this chapter, frameworks relevant for the HMEC approach were introduced and discussed. The selection of frameworks was inspired by their novelty and scientific significance. Analysing these frameworks shows the following issues very clearly.

Existing frameworks are constrained regarding their operational environment. Most of the frameworks are tailored to specific operating systems or even specific operating system versions. The current diversity on the mobile device market is not reflected in the representative set of offloading frameworks. Every single framework targets the Android operating system; not a single framework is capable of offloading on iOS devices, which also have a remarkable market share.

From a security point-of-view, most of the frameworks do not consider security aspects at all. This is impacted by the fact that a majority of the existing frameworks do not provide a sufficient level of flexibility and do not react to changing environments in a dynamic way. mCloud, as the only framework with a dynamic computing resource allocation strategy, also requires an equally dynamic approach of assessing the trustworthiness of the available devices. For frameworks with a static binding to their offloading computing units, a pre-established trust relationship to computing units is sufficient.

Considering the current state of existing frameworks, they are not ready to operate on sensitive data or perform sensitive computations. In the following chapter a security assessment framework is developed which enables to perform a systematic evaluation of the current state of existing frameworks as well as their potentials and provides guidelines towards privacy and data security aware extensions.

Chapter 6

Security Assessment

The state-of-the-art in the field of MCC, HMCC, and HMEC does not contain a solid security baseline. Existing approaches put their efforts on enabling technologies, leaving security aspects behind. In a broader context, distributed computing is not a recent innovation but clearly differs from the concepts of MCC, HMCC, and HMEC. Therefore, a step back is necessary and requirements need to be derived from related and enabling technologies.

In this chapter, a set of security-related requirements is derived in Section 6.1 from security-related research in the topic of cloud computing and grid computing, acting as key enabling technologies. Adaptations are required to reflect the changed context of the technology. Furthermore, MCC, HMCC, and HMEC demand the definition of security-related requirements from two perspectives: from a user's perspective to protect sensitive user data and computations, and from a resource provider's perspective to protect its offered resources. With the derived catalogue of security-related requirements the introduced frameworks from Chapter 5 are evaluated regarding their adherence to this catalogue in Section 6.2.

6.1 Security Assessment Catalogue

In this section a requirement catalogue is defined which enables, by assessing the adherence of frameworks to the requirements, to determine the data privacy awareness and suitability to execute computations on sensitive data. Requirements are derived from literature for related technologies and are based on a threat model analysis. This results in requirements where parts are very similar to the requirements of general cloud computing; others can be traced back to traditional IT systems. This section starts by identifying important assets and threats of HMCC systems and continues countering these threats by defining requirements to mitigate these threats, also by pulling requirements from related researches into the context of HMCC.

6.1.1 Assets and Threats

Researches by Rong, Nguyen, and Jaatun [2012] and Ryan [2013] identify general cloud computing security challenges, also relevant for the more generic context of distributed computing:

- *Access to data:* The cloud provider usually has full access to the data, hence full responsibilities regarding integrity and confidentiality.
- *Location of data:* Big players are acting globally, operating their data centres around the globe. Data and computations can be moved by the operator, according to certain agreements, freely. Cloud providers optimise their operations according to costs of operation. Within the agreed borders, users usually do not have tracing capabilities.

- *Multi-tenancy:* Generally, cloud services are built for scalability, therefore are targeted to serve multiple users and user groups. Users need to trust the cloud provider to have protection mechanisms in place to isolate different users and their data from each other.
- *Monitoring and logging:* Logging is an important aspects of every service. Especially with the growing amount of applications migrated to the cloud, operators require up-to-date logging information. This goes hand in hand with multi-tenancy: Log output will likely also contain sensitive information, which again stresses the need for decent isolation between different user's and different service's data.
- *Cloud-related standardisations:* Standardisations gain popularity. They enable users to assess different aspects of cloud computing offers. On a technical level, a majority of the providers only support applications which are developed particularly for one specific platform (vendor lock-in).

Armed with this background knowledge of cloud computing and the already broad understanding of MCC and HMCC systems the assets can be identified. All further considerations are conducted with the following assumptions in mind:

- **AS1:** The user's device is assumed trusted, otherwise no guarantees can be made.
- **AS2:** The communication channels are authenticated, integrity protected and do not reveal data to third parties.

HMCC does not follow a strict client-server model, still for each interaction a clear separation can be determined. Therefore, a separation between client and server assets is appropriate (client assets are marked with **C**, server assets with **S**):

- ***A1C*** - *Sensitive data:* Data containing private or confidential parts can be considered as sensitive data. Operating on sensitive data is fundamental for most applications and is transported to remote computing units during the offloading operations.
- ***A2C*** - *Sensitive computation:* Computations that result in sensitive data, or computations with a sensitive computation logic, are considered as sensitive computations. In the course of offloading computations, also sensitive computations are offloaded to remote computing units.
- ***A3C*** - *Availability of offloading services:* This asset refers to the availability of the offloading services to the users. The offloading mechanism only provides added value if computing units are available and reachable.
- ***A1S*** - *Availability of provided resources:* Computing units offer different types of their available resources like computational power, storage, or specialised hardware. The available resources, regardless of their types, are limited.
- ***A2S*** - *Resource integrity:* The offered resources are used to process the offloaded tasks. The integrity, in this case, refers to the correct behaviour of the resources, e.g. saved data has not been modified, or computational results match locally performed computations.
- ***A3S*** - *Sensitive computational results:* Results of computations again may contain sensitive data, regardless of the input data's sensitivity.
- ***A4S*** - *Logging data:* Monitoring data produced during processing offloaded tasks may contain sensitive data and information not meant for disclosure.

Analysing these assets reveals the following threats/attack scenarios. A mapping of threats to the corresponding assets is provided in Table 6.1.

- ***T1*** *- Fraudulent computing unit provider discloses sensitive data and computations:* A fraudulent provider is operating a computing unit and has access to offloaded sensitive data, as well as sensitive computations. The provider is able to disclose the sensitive data and computation logic.
- ***T2*** *- Infected computing unit discloses sensitive data and computations:* A computing unit is operated by an honest provider, but the infrastructure has been infected by an attacker. The attacker has access to offloaded sensitive data, as well as sensitive computation logic and is able to disclose this information.
- ***T3*** *- Offloading of malicious code to disclose sensitive data and computations:* A fraudulent user offloads malicious code to a target computing unit and gains the ability to leak sensitive data and computation logic of other users/applications running on the same computing unit.
- ***T4*** *- Loss of Internet connectivity:* Without, or with bad connectivity to the offloading computing units, the availability of the offloading services cannot be provided.
- ***T5*** *- Attacker powers off single connection point:* By relying on single connection points, attackers can negatively affect the availability of the provided computing resources and the general availability of the offloading services.
- ***T6*** *- Resource over-consumption by fraudulent code:* Fraudulent offloaded code may over-consume the provided computing resources and negatively impact their availability for other users.
- ***T7*** *- Fraudulent computing unit provider compromises resource's integrity:* A fraudulent resource provider has compromised its provided resources' integrity. No resource integrity guarantees can be made.
- ***T8*** *- Resource's integrity is compromised due to infection:* A computing unit is operated by an honest provider, but the infrastructure has been infected by an attacker. The attacker gains the ability to compromise the integrity of the provided resources.
- ***T9*** *- Fraudulent computing unit provider discloses computation results:* A fraudulent provider is operating a computing unit and has access to sensitive computation results. The provider is able to disclose this data.
- ***T10*** *- Infected computing unit discloses sensitive computation results:* A computing unit is operated by an honest provider, but the infrastructure has been infected by an attacker. The attacker has access to sensitive computation results and is able to disclose this information.
- ***T11*** *- Untrustworthy administrators with access to log files:* Untrustworthy computing units administrators with access to log files are able to disclose potentially sensitive information contained in log files.
- ***T12*** *- Compromised computing unit gives attacker access to log files:* A compromised computing unit enables an attacker to gain access to potentially sensitive information. The attacker may disclose this information.

Backed by this analysis, the next chapter 6.1.2 continues on defining countermeasures, to be available in sustainable frameworks.

6.1.2 Security Requirements

From the identified threats and security challenges of cloud computing, the security requirements catalogue is derived. The derivation for HMCC/HMEC and MCC systems is based on the identified security

	A1C	A2C	A3C	A1S	A2S	A3S	A4S
T1							
T2							
T3							
T4							
T5							
T6							
T7							
T8							
T9							
T10							
T11							
T12							

Table 6.1: Threat-Asset assignment

requirements of Iankoulova [2011] and Höner [2013], composing an in-depth literature review of available scientific material, and the previously discussed cloud computing security challenges. The requirements are intentionally kept on an abstract level to avoid influencing the technical realisation in a later stage. To a certain extent, the cloud computing security requirements also apply to HMCC/HMEC systems. The definitions are changed to fit into the HMCC/HMEC context, requirements are added where gaps are identified.

For the HMCC/HMEC use case, the cloud computing environment needs to be extended. HMCC/HMEC does not only utilise CC-provided resources, but also computing units in the user's proximity (other mobile devices or cloudlets), and computing units under control of the user (home computers). From a device's perspective, these resources can be seen as another type of cloud resource, but without any prior knowledge regarding the operator. Due to these similarities, a majority of the cloud computing related security requirements as defined by Höner [2013] and Iankoulova [2011] also apply to HMCC/HMEC systems without any changes:

- ***R1*** - *Access Control:* In the context of HMCC/HMEC, access control mechanisms can be considered as the enabler on a technical level, for organisational protection mechanisms. Resource providers need secure ways to limit the users to a certain set of authenticated users. This should not limit resource providers to offer open computing units, but enables the operation of private computing units limited to groups of users with a trust relationship with particular providers. Although this is not a technical solution to counter fraudulent providers or infected computing units, it enables the usage of pre-established trust relationships.

- ***R2*** - *No single-points-of-failure:* A high availability can only be provided in the absence of single-points-of-failure. In MCC or similar infrastructures, a prevalent single-point-of-failure is the connection to the offloading computing unit. Frameworks should not exclusively rely on the availability of Internet connectivity, where possible. Furthermore, single e.g. coordination servers need to be avoided.

- ***R3*** - *Usage of multiple communication technologies:* To increase the fail-safety, frameworks should use different available communication technologies. This includes the Internet, but also local technologies considering the surrounding of the user like Wifi and Bluetooth.

- ***R4*** - *Resource's integrity assurance on technical level:* Integrity in general refers to data consistency over its whole life cycle. In the context of HMEC, computing unit integrity means that

computing units execute offloaded code without applying any modifications, and return the correct result. The resource provider could modify the offloaded code unnoticed by the user. The modified code could then introduce a different behaviour or return erroneous results. As modifications could be applied remotely they are hardly detectable. To mitigate the risk of compromised resource integrity on a technical level, attestation methods need to be included in HMEC solutions, to remotely assure that a given computing unit is in a known good state and operating as expected. These assurance methods mitigate the risk induced by fraudulent providers, as well as due to the infection by other attackers using malicious code.

- ***R5*** - *Complete task isolation:* In cloud computing, storage is often used to backup or synchronise users' data between multiple devices. This also involves sensitive data. In a HMEC context, the cloud providers generally do not deal with this kind of data. However, computing units execute parts of an application that potentially contains sensitive data or computations. It is more complex to extract relevant data, but the data potentially also contains highly sensitive information.

  Depending on the used offloading approach, the system per se has different isolation levels between different users. For virtual machine based approaches, where each virtual machine is dedicated to a single user or even a single application instance of a particular user, high isolation levels can be provided without further measures. The attacker would need to find a security flaw in the virtual-machine hypervisor. To better utilise the available computing resources, a system may share a single virtual machine and execution environment across different users and applications. Without applying further protection mechanisms like sandboxing, data and computations may be accessible to different users and applications operated in the same environment. These approaches mitigate the risk imposed by the offloading of malicious code to disclose sensitive data. Furthermore, it complicates the disclosure of data in the case of infected computing units.

- ***R6*** - *Non-repudiation:* By providing the non-repudiation property, a client cannot deny that it issued a particular message, and the computing units can be sure about the issuer. Only the issuer is in charge of the required private information to produce authenticated messages. In the HMEC context it is important that computing units can unambiguously link requests to particular issuers. Computing units may expose sensitive information, as a result of sensitive computations, to the clients. This mitigates the risk of exposing sensitive data to unauthorised users.

- ***R7*** - *Protection against resource over-consumption:* Providers need to protect their provided resources by either analysing the submitted tasks, or by rendering the execution unattractive for malicious user (e.g. applying a cost model).

The effectiveness of the defined countermeasures on the identified threats are defined in Table 6.2.

	T1	T2	T3	T4	T5	T6	T7	T8	T9	T10	T11	T12
R1	■	■	■				■	■	■	■	■	■
R2				■	■							
R3				■	■							
R4	■	■	■				■	■	■	■		
R5		■	■							■		
R6			■	■		■						
R7						■						

Table 6.2: Requirement-Threat assignment

The seven security requirements defined in this section will be used to systematically assess security capabilities of the introduced HMCC frameworks.

6.2 Evaluation of Existing Frameworks

In this section, the security assessment catalogue as defined in Section 6.1 is evaluated against the introduced, existing frameworks. For classic cloud services, where cloud providers are used as storage providers, properties like integrity and confidentiality can be guaranteed by utilising client-side encryption and protection mechanisms, availability on the other hand is achieved using replication techniques. In case the cloud service does also process data, the situation gets more complex. One promising approach to meet defined security requirements is the use of fully homomorphic encryption. This cryptographic concept enables operations on encrypted data. Other solutions require the provision of the used encryption key to the cloud-service provider, providing no gains in security.

HMCC/HMEC is completely focused on data processing. Therefore, additional measures are required to fulfil the defined security requirements. Hence, the capabilities of current HMCC frameworks are assessed regarding their adherence to the security requirements catalogue. A summary of this assessment is provided in Table 6.3. The table encodes assessment results with the help of three different symbols: "✓" for requirements which are completely fulfilled, "✗" for requirements which are not fulfilled, and "∼" for requirements which are partly fulfilled or could be fulfilled based on the documentation. If table cells are left blank, the requirement has no relevancy for the framework.

6.2.1 R1 - Access Control

Applying access control, computing units can control which clients are allowed to use the computing unit. Granting or denying access could be based on one or multiple of the following properties:

- User's identity
- Device's identity
- Device's location
- Device's assessed trust level

These properties have a dynamic character, hence may change during operation. This requires computing units to dynamically evaluate access requests and react to changing environments. The introduction of the frameworks shows a clear focus on technological aspects and improving the state-of-the-art regarding the offloading techniques. Most of the frameworks have a static binding to their computing unit, initialised at deployment. Although this approach could be considered as access control, it fails in adhering to basic dynamic requirements. The access control specification at the computing units should be detached from any program logic and should enable referencing the mentioned parameters and beyond.

6.2.2 R2 - No single-points-of-failure

Reliability can only be provided by systems that do not have a single-point-of-failure and feature fallback mechanisms. MAUI, CloneCloud, and TANGO perform a static binding before the application is enrolled to the target devices. This represents a completely static approach and does not consider the dynamic nature of mobile operational environments.

Cuckoo is, conceptually, not bound to a single computing unit. A resource manager keeps track of available computing units and assigns tasks to suitable computing units. Still, the setup and registration of computing units at the resource manager is a manual process. ThinkAir separates the available computing units in a primary server and secondary servers. The primary server keeps track of available secondary servers and their utilisation. A primary server dynamically starts and stops secondary servers where the offloaded tasks are executed. Hence, without the primary server, the whole system is not usable. mCloud

follows a similar approach regarding its remote cloud resources. A central registry is setup, which keeps track of all available computing units. Clients can acquire computing units from the registry. This suffers from the same issues than ThinkAir's approach. Without a registry, the system cannot be used. Besides discovering resources from the central repository, mCloud also discovers nearby computing units like cloudlets and other mobile devices via Wifi technology. mCloud is the only existing approach utilising different communication technologies, still, the registry acts as a single weak point interfacing with the computing resources.

The key to reliability is the avoidance of single-points-of-failure. If a system totally relies on central registries or control instances, the reliability of the whole system is compromised by single weak links.

6.2.3 R3 - Usage of multiple communication technologies

The usage of multiple communication technologies enables to increase the availability of the system. All frameworks, except mCloud, exclusively rely on Internet connectivity, therefore their operability depends on the availability of a proper Internet connectivity. mCloud also considers nearby devices, eliminating the communication channel as the single-point-of-failure.

In a HMEC environment, the following strategy is pursued to increase reliability, independent of available Internet connectivity:

- Provide functionality over the Internet when connectivity to powerful remote resources is available.
- Provide functionality by exploiting nearby devices with local communication means (WiFi, Bluetooth) to avoid service unavailability in offline scenarios.

Assuring full operability in both scenarios precludes the use of a central and coordinating server. To assure full operability in offline scenarios, the resource-constrained device needs to be aware of available resources at any time during execution. This, in turn, increases complexity for the local device.

6.2.4 R4 - Resource's integrity assurance on technical level

The requirement for integrity refers to the execution of unmodified offloaded code. In fact, users cannot verify if the correct code is executed on the remote computing unit. Only the TANGO framework slightly considers this issue. Execution in TANGO does not follow a partitioning-based approach but uses process replication. The framework is not limited to two replicas and can, in principle, be extended to an arbitrary number of replicas, acting as fail-safe replicas. All replicas can then consent on one result, ruling out biased results. Currently, the framework and its evaluation are focused on user-perceived latency and energy usage, fail-safety is more considered as a theoretical concept.

None of the other frameworks considers this issue. The solution of TANGO is specific to the framework. A user has two ways to mitigate the risk of using dishonest remote computing units. Either the user assures to be in sole control of the computing unit, or special means are utilised to assure the trustworthiness of the resource and its operator. Technical solutions include remote attestation procedures that enable the state verification of computing units. Other approaches could include the execution of the offloaded parts in trusted areas.

6.2.5 R5 - Complete task isolation

In contrast to the requirement for integrity, isolation focuses on the data and sensitive information provided by the user. Task isolation is not directly tackled by the existing offloading frameworks but has a high dependence on the utilised offloading method and utilisation of computing resources by multiple users. If computing units are exclusively bound to mobile devices or application instances, the potential

of leaking data is minimised. This especially is the case for CloneCloud and ThinkAir. This does not foster an ideal resource utilisation but enables optimal data protection. Frameworks utilising late and dynamic computing unit binding require further considerations. None of the examined frameworks, besides mCloud, feature a late and dynamic computing unit binding. The mCloud framework does not take any considerations in this regard.

6.2.6 R6 - Non-repudiation

For productive services, the non-repudiation property is important to not reveal sensitive data of one user to another. This issue is not directly tackled by any of the described frameworks but again is only relevant if multiple users refer to the same computing unit. All frameworks, besides mCloud, exclusively bind their computing units to mobile devices, therefore non-repudiation for dynamic environments is out of their scope. Also mCloud, with its proximity enabled approach, does not feature methods to ensure the non-repudiation property.

6.2.7 R7 - Protection against resource over-consumption

All requirements discussed so far mainly tackled issues affecting users and the security of their personal data. This requirement, in contrast, focuses on the offered resources of providers. Regardless if frameworks are statically bound to computing units or linked dynamically, in both scenarios users may offload tasks and execute fraudulent activities, or may consume all available resources to their full extent. Again, due to the prevalent static binding of existing frameworks and putting the focus on offloading methods, protection mechanisms against over-consumption are not considered so far. Analysing this requirement reveals that the fulfilment can take place on different levels:

- On a technical level one approach to prevent misuse of provided computing units is the analysis of offloaded tasks. Still, this is deemed to be an computationally intensive task.
- On an organisational level, users could establish trust relationships with each other and with providers on the one hand and make misuse unattractive for attackers by e.g. requiring micro payments for resource utilisation on the other hand.

Table 6.3: Security requirements evaluation of existing frameworks, using the following symbols: "✓" for requirements which are completely fulfilled, "✗" for requirements which are not fulfilled, "~" for requirements which are partly fulfilled or could be fulfilled based on the documentation, empty fields indicate that a requirement does not apply to the corresponding framework

	MAUI	CloneCloud	Cuckoo	ThinkAir	TANGO	mCloud
R1	✗	✗	✗	✗	✗	✗
R2	✗	✗	✗	✗	✗	~
R3	✗	✗	✗	✗	~	✗
R4	✓	✓	✓	✓	✓	✗
R5						✗
R6	✗	✗	✗	✗	✗	✗
R7	✗	✗	✗	✗	✗	✗

Chapter 7

Part II Conclusions

This part of the thesis starts with the introduction of several offloading frameworks. Various approaches are available to the scientific community, but only a fraction provides novel and scientific relevant contributions. The relevancy of frameworks is assessed by their novelty and scientific contribution at the time of publishing. This selection procedure results in a list of six frameworks, examined in detail. In parallel, a security assessment catalogue is produced to evaluate the sensitive data-awareness of HMCC/HMEC frameworks. The development of the catalogue follows well-established methods of defining threat models including security requirements. For the derivation of requirements, a practical relevance is ensured by logically deriving them from related technologies like cloud computing and distributed computing in general. The resulting catalogue is a tool for security assessors to evaluate frameworks and guides developers through the process of specifying and designing new solutions.

The evaluation of the security requirements catalogue on the existing frameworks reveals that a majority of the frameworks focuses on technical and implementation aspects like partitioning or offloading techniques. Data security in HMCC/HMEC systems is still in its infancy and requires in-depth considerations. Furthermore, the analysis reveals that all frameworks are locked to specific execution environments, not considering interoperability and cross-platform aspects. Additionally, a majority of the frameworks misses offloading opportunities by adhering to a strict computing unit selection process and not considering nearby computing units for a more dynamic and flexible setup.

These three major discoveries, the lack of interoperability, flexibility and the missing focus on data security, is the driving force of this thesis. The consequences of these discoveries manifest for applications operating on sensitive data and especially corporation controlled devices with applications operating on sensitive data. Corporations require complete control of the data flows. Parts of their data may only be processed on-site, others may also be processed by trusted external computing units. Frameworks that cannot provide security guarantees hinder the usage in these scenarios. Still, corporate environments offer many opportunities with large fields of computationally intensive applications to be exploited by HMEC solutions. The advancements achieved by the HMEC technology manifest in a progressive penetration of mobile devices, hence increased flexibility and computational capabilities, also in the daily work on sensitive data.

The contributions of this part, the threat model and security requirements catalogue, can also act as the baseline for other developments in this field of research. They can help in the analysis and improvement of the data security awareness and privacy protection mechanisms of existing frameworks in the same way than it is applied in the following chapters. The outcomes of this part steer the further work on this thesis and codetermine the structure. The remaining main contributions of this thesis pick up the identified prevailing weaknesses in interoperability, flexibility and data security and enhance the state-of-the-art. In Part III the focus is put on the interoperability and flexibility issue. A more flexible architecture is required to respect flexibility requirements from scratch, already on an architectural level. This architecture requires anchors, where different data security enabling approaches can attach.

The enhanced architecture and model is derived from the findings and security requirements catalogue in this part. Although the architecture has major differences to previous approaches, a broad reusability of already existing components is achieved. Isolated technical aspects like the offloading and partitioning approaches, as well as the integration in the application, are not reinvented and are reused to a high degree. In practice, the different available approaches can equally be combined with the new architecture.

The interoperability issue cannot be solely solved on the architectural level. The implementation of the proposed architecture in Part III goes beyond a concept implementation and takes new paths. The interoperability issue is solved by focusing on technologies which are available on all major mobile devices' operating systems and architectures. Compared to native techniques, these approaches are constrained in their capabilities. Ways need to be developed to overcome these barriers.

Data security and privacy protection mechanisms are then detailed in Part IV. They seamlessly plug into the architecture and increase the offloading flexibility. Techniques are developed and integrated to enable the usage of third party devices in sensitive data processing use cases. Furthermore, explicit control of data flows by data owners is enabled by integrating with policy environments.

The overall outcome of this thesis is a novel processing model: Hybrid Mobile Edge Computing (HMEC) including a reference architecture and implementation, founded on a solid threat model and designed according to the security requirements. The solution can act as a baseline for future researches and developments in this sector.

Part III

A Novel HMEC Approach

Chapter 8

A Re-usable HMEC Architecture and Solution

People are more and more migrating their daily work from desktop computers to mobile devices. The usage of mobile devices first outranked desktop systems in 2011[1] with the effect that more complex and energy intensive applications are executed on mobile devices. This results in users demanding extended battery lifetimes. Currently deployed solutions are MCC solutions where distant clouds offer services for applications to perform their pre-defined tasks: Save data in the cloud to virtually extend the device's local storage, upload an image for processing, calculate routes, or whatever one can think of.

This approach always suffers from high latencies, compared to local resources. Fernando, Loke, and Rahayu [2012] highlight that a close collaboration between proximate devices could enable new use cases and make existing use cases more efficient due to low latency connections:

- *Image processing/natural language processing:* As an example, the authors use an OCR operation which requires a lot of processing power. The mobile device is not capable of performing the operation. The user could connect to a remote server or search for nearby devices interested in participating in the calculation. There may also be other devices nearby, interested in the same image being processed. Using the remote server might be subject to high roaming costs and high latency.
- *Crowd computing:* Combine the data from multiple devices nearby to get a complete view. The authors take the example of video captures from multiple angles of the same event, resulting in a video with different perspectives.
- *Sharing GPS/Internet data:* To save energy, a group of nearby people could share their GPS data, so only a single device needs to have GPS activated. The same applies to data transferred from the Internet. If multiple nearby devices request the same piece of data, only one needs to download it and can share it locally.

Currently, existing frameworks are very restricted in this regard. They interpret MCC only as a tight collaboration between mobile devices and distant cloud services. Thus, they miss use case opportunities and are not as efficient as they could be. In this chapter, an architectural view of a reusable HMEC solution is proposed. After defining the requirements on a reusable HMEC architecture, an abstracted view on the current state of architectures from well-known and analysed frameworks is presented.

The methodology followed in this chapter is defined as the following:

- First, identify the strengths and weaknesses of existing architectures.

[1] http://www.canalys.com/newsroom/smart-phones- overtake-client-pcs-2011

- Second, combine the strength and existing approaches with new technologies to eliminate the drawbacks.

- Third, evolve an architecture which is not targeted at specific frameworks.

8.1 Current Approaches

A majority of the existing MCC frameworks and solutions focus on the issue of *what* should be considered for offloading under *which* conditions and *how* the offloading should be performed technically. The question of *what* to offload under *which* conditions is answered multiple times, resulting in different partitioning models as introduced in Section 8.1.1. The technical details about *how* to perform the offloading process also results in a few models as illustrated in Secion 8.1.2.

8.1.1 Partitioning Models

Partitioning models provide means to determine which parts of an application should be executed in an offloaded mode and which parts should be executed locally. The models can make use of offline analysis before the application is deployed or on-the-fly during the execution of the application. Generally, on-the-fly approaches offer increased flexibility, whereas offline approaches are more static and are generally unable to adapt to changing environments.

An automatic offline application-partitioning model is introduced by Chun et al. [2011] targeted at minimising total execution time or at minimising the energy consumption on the mobile device. An offline analyser inspects a binary application and identifies candidates for migration and re-integration points in the application. Their analyser adheres to three fundamental properties:

(a) Methods that access specific features of a machine must be pinned to the machine.

(b) Methods that share native state must be collocated at the same machine.

(c) Nested migration must be prevented.

The profiler then profiles the application methods with random input data on the mobile device and on the computing unit in the cloud. The optimisation solver decides which method calls should be offloaded, based on the collected data for each supported optimisation profile (e.g. optimise energy consumption or optimise application run-time). From an implementation perspective, using this approach additional instructions are inserted into the binary to represent the migration and re-integration points. These instructions need to be handled by the operating system, which might require modifications.

A different approach is introduced by Cuervo et al. [2010]. They follow an online approach where the initial partitioning is performed by the application developer. The developer annotates methods which should be considered for offloading during the runtime of the application. At runtime, the profiler decides if offloading is beneficial based on three factors: (1) smartphone's energy consumption characteristic, (2) runtime and resource needs of the particular method, and (3) network characteristics such as bandwidth, latency, and packet loss. Estimating the runtime of methods is based on the runtime of past executions. The solver then uses the collected data to solve the optimisation problem that determines which method should be executed locally and remotely, comparable to the optimisation solver introduced in the former approach. From an implementation perspective, this approach does not require awareness of the operating system, it can purely be handled by the application and the offloading framework.

8.1.2 Offloading Techniques

Offloading techniques refer to the means used to (a) migrate the execution of an application to a remote computing unit, and (b) to the concrete execution model used on the remote side. Reiter and Zefferer [2015a] refer to three distinct offloading approaches:

1. *Virtual Machine Based Approaches*

   Virtual machine (VM) based approaches, as described by Chun et al. [2011], aim to provide a runtime environment, which is similar and compatible to the environment of the resource-constrained device. Following this approach, the VM can execute unchanged mobile-device applications, lowering the required development effort and enabling a fast and broad rollout. The disadvantages of this approach are obvious. First, a vast amount of different virtual machine images reflecting different hardware and operating-system combinations is required. Second, for efficiency reasons, a VM will only be started when required, which results in a few seconds delay for the user. Third, all applications need to be available at the resource provider's side or need to be uploaded, which again results in a delay. Finally, the VMs consume a non-negligible amount of resources, which may even exceed the resources required by the application. In general, VM-based approaches are hence inappropriate in most cases.

2. *Class- or Method-Based Concepts*

   As described by Cuervo et al. [2010], class or method based concepts provide a more fine-grained approach and focus on the offloading of classes or methods instead of entire applications. Before methods are invoked, it is decided, if the method is executed locally or remotely. Depending on the utilised programming language, it may not be required to provide a compatible execution environment. Using managed programming languages (for example the Java programming language or the .NET environment), it is possible to transfer the current state of an application to another machine, which may be running a different operating system. Class or method based concepts require more support from the developer, but in return eliminate delays induced by the VM-based approach. These concepts might also require the mobile device to upload parts of the application to the offloading target. However, by using smart extraction of the relevant application parts, it is possible to minimise the size of these data.

3. *Object-Based Concepts*

   Object-based concepts as proposed by Sinha and Kulkarni [2011] extend the class-based approach to an even more-fine grained level. Sinha and Kulkarni [2011] introduce object-based concepts by means of the concrete example of an image-manipulation software, where a certain class is used to apply different effects. With a class-based approach, this class would always be offloaded to the same target, or would always run locally. Utilising the object based approach, the system bases the offloading decision on the prediction of the required computational effort for one specific instance of a class, i.e. an object. This way, different instances of the same class can be offloaded to different targets.

[Reiter and Zefferer, 2015a]

A recent novel approach relying on process replication is introduced by Gordon et al. [2015]. It provides a completely different execution model and is listed here for completeness. Following this model, an application is executed on different replicas, with one leading replica displaying results to the user. The whole data is generated on each replica, therefore communication effort is minimised. Still, this approach does not fit to the overall goal of this thesis to reduce energy consumption.

Table 8.1: Overview of utilised offloading technique of the analysed frameworks

	VM-based approach	Class- or method-based approach	Replication-based approach	Object-based approach
MAUI	·	✓	·	·
CloneCloud	✓	·	·	·
ThinkAir	✓	·	·	·
Cuckoo	·	✓	·	·
TANGO	·	·	✓	·
mCloud	✓	·	·	·

Virtual machine-based approaches require the least integration effort in applications, and may even be applied to unchanged applications. Nevertheless, the approach relies on the availability of virtual machines with nearly equal characteristics as the client device. Furthermore, it requires a vast amount of resources to start one virtual machine per client device, or even one virtual machine per client application. Object-based concepts, on the other hand, are able to apply a fine-grained offload control mechanisms on source-code level. This seems more flexible than the method-based approach. Actually, by introducing a decision engine, which can decide on a call-by-call basis if offloading should be performed, the method-based approach becomes even more flexible and provides even more fine graininess.

Looking at the existing frameworks reveals that most of the existing frameworks are based on the virtual machine-based approach as highlighted in Table 8.1. As discussed above, virtual machine-based approaches require vast amounts of memory to represent a majority of the available devices and their different configurations. This drives the need to research in other directions and discover other, more efficient approaches.

8.1.3 Current Architectures

As described by Reiter and Zefferer [2016] and based on the findings in this chapter, general building blocks and a general architecture is extracted. The architecture maps to the overall functionality of the analysed frameworks, some peculiarities may not be covered without impacting the functionality of the frameworks. The architecture is shown in Figure 8.1. It is separated in two parts, the offloading client framework, executed on the client-side, and the remote side. Still, the focus of most implementations clearly lies on the client-side. Frameworks differentiate between four components:

- The *Profiler* provides performance values for different contexts. An *Application Profiler* is used to estimate the runtime of the currently analysed part of the application. Application profilers may operate online, while the application is actually executed, or offline, where performance values are calculated in advance. The *Context Profiler* provides performance values for the current execution context. Depending on the used framework, the context profiler may gather values like available

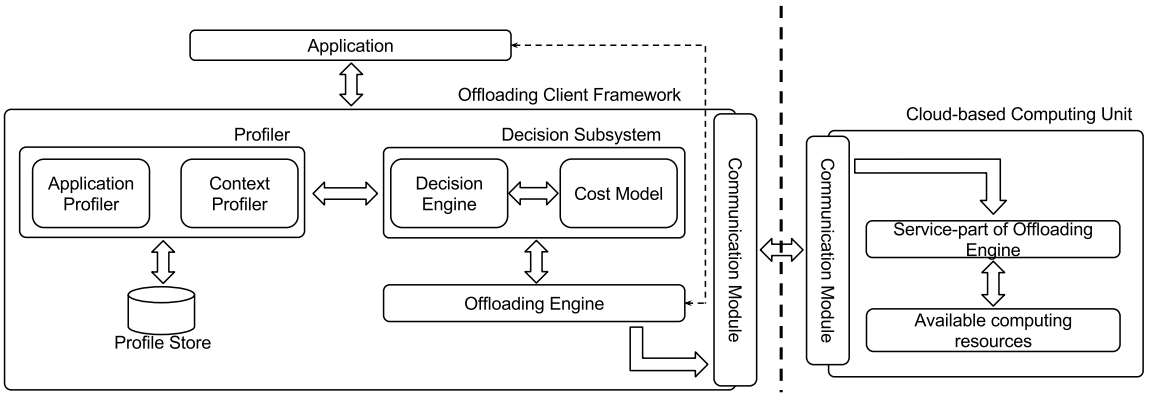


Figure 8.1: Current MCC Architectures

bandwidth, latency to the remote computing resource(s), battery status, or current CPU utilisation. A *Profile Store* is attached to the profiler where results can be stored for later reference.

- The *Decision Subsystem* involves a *Decision Engine* and a *Cost Model*. The goal of the decision subsystem is to get a final choice whether a method call should be offloaded or executed locally. The decision engine gathers the performance values from the profiler. These values are applied to the cost model to get a final decision. The definition, if an offloading operation is beneficial, is done by the cost model. Cost models can focus on increasing performance, on minimising energy consumption on the mobile device or provide a well-balanced variant as a combination of both.

- The *Offloading Engine* performs the actual offloading operation on a technical level. Depending on the utilised offloading technique, as described in Chapter 8.1.2, the offloading engine needs to perform synchronisation of the local and remote environment. Finally, control is transferred to the remote side. The results are re-integrated in the local application when available.

- The *Communication Module* acts as the key component in communication with the remote side. It establishes the connection to the computing unit used for the offloading operation and enables the transfer of offloading requests, as well as the reception of responses.

These components are coordinated to enhance the user experience of mobile applications. Nevertheless, the existing approaches have major issues in security, flexibility, efficiency and interoperability. The security of existing frameworks is analysed in Chapter 6. Flexibility and efficiency are closely related. As Figure 8.1 illustrates, the communication module communicates with a single cloud-based computing unit (or pre-defined set of CC units). If no connection to this computing unit is available, the whole system does not work anymore.

In the CloneCloud approach [Chun et al., 2011] the cloud computing unit is statically bound to the application at compile time. In the case of bad connection conditions to the computing unit, the framework can advice to not execute remotely and is stuck with local execution. Similar considerations apply to ThinkAir [Kosta et al., 2012], although it is more dynamic in terms of allowing parallel execution in multiple VMs, there is still a single master-cloud service as a single-point-of-failure. Cuckoo [Kemp et al., 2012] tries to remove these limitations by introducing a resource manager, which distributes the work on different workers, deployed at different locations. Still, manual registration is required. Goyal and Carter [2004] consider the fact that surrogates could be run on different types of devices, but still rely on VM technology. The ultimate drawbacks of full-fledged device VMs are the high memory requirements and long setup times. The size of a single device VM starts at hundreds of megabytes and can go up to several gigabytes. Furthermore, device VMs need to be close to the feature-set of the original device in terms of used operating system version, execution environment, or supported libraries. Ha et al. [2013] try to counter this issue by proposing a dynamic VM synthesis for dedicated devices and use cases. The

results of their work show that synthesising a dedicated VM takes up to several seconds or even minutes. In mCloud [Zhou et al., 2015] a hybrid concept is introduced where also local computing units are considered. For cloud offloading, Android VMs are started in the cloud, for offloading to other devices, offloaded code is directly executed on the Dalvik VM on the target device.

Beside the mCloud approach, which breaks out of the described architecture in terms of connectivity, all frameworks rely on static connections to their (mostly) cloud-based computing units. This works great where reliable Internet connections are available, but a constant and fast connection is required. As Cuervo et al. [2010] highlight, increasing the RTT from 10ms to 25ms for code offloading operations with 10kB almost doubles the energy consumption. With increasing RTT the energy consumption increases linearly. Definitely, mobile Internet connectivity is getting better, but still does not fulfil the strict performance requirements for HMCC/HMEC frameworks. A study by Horsmanheimo, Maskey, and Tuomimäki [2014] concerning the feasibility of utilising Universal Mobile Telecommunication System (UMTS) and Long Term Evolution (LTE) for smart grid applications reveals that UMTS is exposed to large latency variations in peak hours with RTT values above 100ms. LTE RTT values are stable at the moment, but the authors claim that this may be due to the still low penetration of LTE.

Another issue is the interoperability of the frameworks between different device-vendors and operating systems. Although not an architectural issue in its nature, it still is impacted by architectural choices. Code offloading of native applications requires a computational unit which can at least execute the offloaded code. The execution environment should be as close to the original execution environment as possible. Most of the frameworks focus on Android-based devices equipped with a Dalvik VM or even require a modified operating system version to catch the migration and re-integration instructions. This rules out all other players like iOS, Windows Phone and smaller initiatives like Ubuntu Phone[2]. Techniques and approaches developed for Android and its Dalvik VM might not be applicable for other environments.

To summarise the findings of this chapter, existing frameworks interpret application offloading only as a tight collaboration between mobile devices and provided cloud resources. Minimising latency is key for offloading frameworks to improve the performance of applications. This is currently hindered by the fact that (a) most existing offloading approaches use virtual machine-based approaches with long setup times and are often strictly connected with a single CC unit. Therefore, existing approaches have clear gaps in terms of flexibility and efficiency. Furthermore, existing frameworks are completely focused on single execution environments and single architectures, with some even requiring modified operating systems versions. Therefore, a clear gap in interoperability exists.

8.2 Design Principles

In this chapter, the overall goals of the architectural enhancement process are defined, based on the previously identified gaps. Besides the security-related requirements, which are mainly considered in Part IV of this thesis, weaknesses were identified in the previous chapter. Furthermore, MCC shows a tight connection to CC, therefore design principles of CC systems can be translated into the MCC context.

As it is the case with CC, MCC operates using different delivery models:

- IaaS in CC can be compared to acquiring a complete external device clone where applications are executed in parallel (compare with CloneCloud from Section 5). This model can be referred to as *MIaaS*.
- PaaS in CC can be compared to source code level application-offloading in MCC, where single, computational intensive parts of an application are offloaded to an acquired computing unit. This model can be referred to as *MPaaS*.

[2] http://www.ubuntu.com/phone

- There is no direct counterpart of SaaS in CC for MCC environments, but the closest are services operated for mobile devices to perform dedicated tasks.

Matching the gaps discovered in Section 8.1 with the introduced delivery models reveals that large parts of the inflexibility come from relying on *MIaaS*. IaaS in the context of CC gives a lot of flexibility and does not constrain the user. It enables the realisation of very different systems and solutions and provides users properties like scalability, pay-as-you-go, and efficiency. IaaS and *MIaaS* require a consistent set of available computing resources. This is not an issue in a CC environment, but is not available in a dynamic and loosely coupled HMCC/HMEC environment. The MCC counterpart for SaaS on the other hand already is too narrowed in its approach. SaaS is targeted at providing a specific solution, serving software solutions from the cloud such as document processing, e-mail applications, or access to online storage.

Therefore, *MPaaS* is the perfect abstraction level to base HMCC/HMEC environments on. CC environments, computing units close to the user, and lightweight devices in the user's proximity can equally offer a platform to be used by applications to offload their computations. This decision basically rules out heavyweight virtual machine-based solutions and enables the usage of lightweight solutions, where only fractions of an application are offloaded.

This, in contrast to the *MIaaS* approach, raises the issue of interoperability. The various available environments like Android, iOS or Windows use different programming environments. An application written in a Java variant, as used for native Android applications, cannot be executed on a device using iOS or Windows. Still, the ultimate goal is to provide a system which makes use of devices, or more generally, of available computing units, regardless of their operating system or hardware architecture. Interoperability beyond operating systems and device architectures enables a deployment and usage on many devices, which drives the adoption rate. With increased adoption rates, scalability is increased in parallel, due to more available computing units.

These observations and choices results in the following requirements on an improved architecture and implementation. The flexibility of MCC systems is increased by not tying offloading frameworks and applications to single cloud-based computing units. Opening different platforms to the system requires a flexible organisation of the available computing units, as well as an offloading approach which is not targeted at single programming languages or architectures. Security is another important aspect, with a separate dedicated Part IV in this thesis. The requirements can be summarised as *Lightweight*, *Interoperability*, and *Flexibility*:

- *Lightweight:* The framework should follow a lightweight offloading approach without requiring a constant set of available computing units.
- *Interoperable:* The framework should enable usage on all major mobile (Android, iOS) and major desktop (Windows and Linux) operating systems, regardless of the utilised hardware architecture.
- *Flexible:* The framework should enable the seamless usage of different computing unit classes and is in charge of deciding whether to use a particular computing unit.

8.3 Evolved Flexible Architecture

In this chapter, the discoveries made in the previous chapters 8.1 and 8.2 are processed to develop an architecture which counters identified flaws on the architectural level but also reuses existing concepts where possible. The targeted implementations are not MCC solutions only, but also more dynamic setups where other computers and also other mobile devices are integrated into the system.

To ensure a clear arrangement and good understanding, the client-side and server-side architectures are separated. The evolved client-side architecture is illustrated in Chapter 8.3.1, whereas the server-side

architecture is illustrated in Chapter 8.3.2. In current architectures, the server-side is often described very loosely. This work focuses on providing a complete solution, focusing on the client- and server-side. To be clear about the designation, the terms client and server are not bound to specific devices in the system. A device acting as the active part initiating an offloading operation is the *client*, the offloading target is the *server*. The terms are only fixed for a single interaction. A device acting as a client in one interaction could act as a server in another interaction. The following chapters describe a more detailed version of the architecture as introduced by Reiter and Zefferer [2016].

In Chapter 8.3.3 different approaches are discussed to interconnect nodes. At first, this might seem like an implementation aspect, but architectural and topological choices already have major impacts on the flexibility of resulting systems.

8.3.1 Client Architecture

In this chapter, an enhanced client architecture is developed, targeting the discovered gaps of existing architectures. This includes the high-level functional requirements from Chapter 8.2 as well as security requirements from Part II. Not all security requirements can be considered on the architectural level and need to be considered in detail in the implementation section and in Part IV.

The requirements for a lightweight and flexible approach are the driving forces to add dynamic approaches in the enhanced architecture. As illustrated in Figure 8.2a and Figure 8.2b, profilers need to follow a more dynamic approach. Current architectures often rely on static analysis phases, performed before the application deployment. This restricts the offloading framework to a limited set of profiles, which need to fit in the current context. This is an appropriate approach for current frameworks. Due to their static binding, the possible cases are limited. Following a dynamic offloading approach also requires dynamically collected profiling data of the executed application. Dynamic refers to information collected at run-time, and later being used to improve the decision process. The dynamic profile store keeps track of collected profiling information at runtime and provides this information to other components in the system to reach a final offloading decision. This does not restrict the static analysis of applications prior their deployment but emphasises the importance of a dynamic component in the decision process.

(a) Current architecture **(b)** Enhanced architecture

Figure 8.2: Profiler enhancements

Continuing on the flexibility requirement, current offloading frameworks and their applications are configured to serve a single designated purpose through the implementation of a cost model as illustrated in Figure 8.3a. Cost models describe the objective of an offloading operation. Currently, the following models are prevalent:

- Using the *Performance Model*, offloaded parts are remotely executed to increase the performance of the offloaded application and the whole client device. The performance model may also parallelise independent tasks to different computing units to simulate a device capable of executing multiple threads simultaneously. The performance model does not necessarily come with energy savings when comparing certain time slots, but only for savings regarding single tasks. Given a

task that takes 200ms to execute directly on the device, and only 20ms on an external computing unit using the performance model. The device can now schedule 10 tasks in the same time compared to device-only execution. Summing up the energy consumptions of the offloading operations may exceed the energy consumption of a single task executed on the device. However, tasks are performed faster, and a better user experience is provided.

- The *Energy Saving Model* on the other hand does not aim at increasing the performance, but only at lowering the energy consumption of an application without impacting the performance. Using this model the computing units used for offloading mimic the performance of the client device. This brings a maximum on energy savings. Again consider a task that takes 200ms to process on the client device. In an offloaded scenario using the energy saving model, the task still takes 200ms, but the device only has to invest some energy for the offloading operations. Again considering the energy consumption of the 200ms time frame, it is decreased to a minimum as only a single offloading operation takes place.

Cost models in today's frameworks use a fixed balancing of these approaches. The decision engine is extended to support multiple cost models as illustrated in Figure 8.3b. Users require a balanced procedure, making use of both approaches with changing requirements over time. The approaches can get weighted based on collected context parameters. One strategy could be, as an example, to balance the performance and energy model with 70/30 as long as the battery is above 50% and then gradually start increasing the impact of the energy model or driving the decision based on user choices.

A further introduced model is *Feature-based Offloading*. Using this approach, the application and offloading framework make use of features not available on the client device but on a dedicated computing unit. Assuming an application that performs operations on smartcards, which are likely not available on mobile devices. One motivation for offloading is to enable operations on particular smartcards, by offloading the methods to preselected and authenticated computing units with attached smartcards. Of course, this requires strong mutual authentication, so both parties can be sure that only authorised client devices can use the smartcard, and client devices can be sure to use the correct smartcard. The computing unit will most likely be operated in a user-controlled environment. This approach can be applied to different problems, involving hardware limitations or attached hardware which is not available for particular classes of devices.

(a) Current architecture **(b)** Enhanced architecture

Figure 8.3: Decision subsystem enhancements

The above enhancements do obviously not contribute to fulfilling the security requirements in the system. As shown in Figure 8.4a and Figure 8.4b, compared to current architectures, a *Trust Mediator* is introduced in the enhanced architecture to fulfil the client-side parts of the security requirements. The trust mediator assists the decision engine in selecting appropriate computing units. Implementations need to introduce a way for developers or need to use tools to determine the required level of trust for a particular offloading request. The trust mediator matches the required trust level with the assessed trust level of computing units. At this level, only a distinction between *trusted* computing units and *untrusted* computing units is made. Trusted computing units could be units which are operated in user controlled environments, whereas untrusted environments could be untrusted devices or untrusted public cloud providers. This distinction is important because untrusted computing units will more likely be available. Not taking untrusted computing units into account at all leaves out many opportunities to improve the

overall performance of applications and constrained devices. On the other hand, it needs to be assured that tasks operating on sensitive data are only processed on computing units trusted by the user. A more fine-grained and extended solution is presented in Part IV of this thesis.

(a) Current architecture **(b)** Enhanced architecture

Figure 8.4: Offloading integration enhancements

The shortcoming of existing frameworks regarding a lightweight and flexible approach are also targeted by the introduction of a background task for the discovery of available computing units, illustrated in Figure 8.5. The component keeps track of available computing units suited for the user's offloading operations. This includes distant computing units operated in the cloud, as well as proximate computing units. The concrete discovery process changes with the utilised interconnection strategy and technology. An optimal implementation always has a set of ready to use computing units, connected via different communication technologies and without a single coordination server or contact point. Under these conditions, the component does not only contribute to providing a lightweight and flexible solution but also contributes to the fulfilment of security requirements:

- *R2 - No single-points-of-failure:* Although the component cannot fulfil this requirement on its own, it provides a major contribution by enabling the distribution of offloading requests to different, already discovered, computing units.
- *R3 - Usage of multiple communication technologies:* The background discovery component provides an abstraction for the available computing units. The technology used to communicate with the computing unit is insignificant. Again, the component acts as an enabler to fulfil this security requirement. Still, contributions from other components are required and need to be aligned.

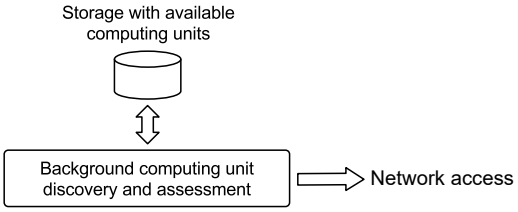


Figure 8.5: Background computing unit discovery

The consolidated client-side architecture, including the communication technology abstraction, is shown in Figure 8.6. The architecture for each of the computing units, acting as a server, is described in Chapter 8.3.2. Besides the already discussed enhancements, the most obvious improvement seen when comparing Figure 8.6 with the abstraction of current architectures in Figure 8.1 is the introduction of a flexible network infrastructure where computing units can log-in and offer their available computing power to other users, or connect to request assistance with computations. Removing the static cloud connection from previous approaches and enforcing a dynamic approach provides more flexibility in resulting systems. Furthermore, the network infrastructure acts as the backbone of the whole framework and

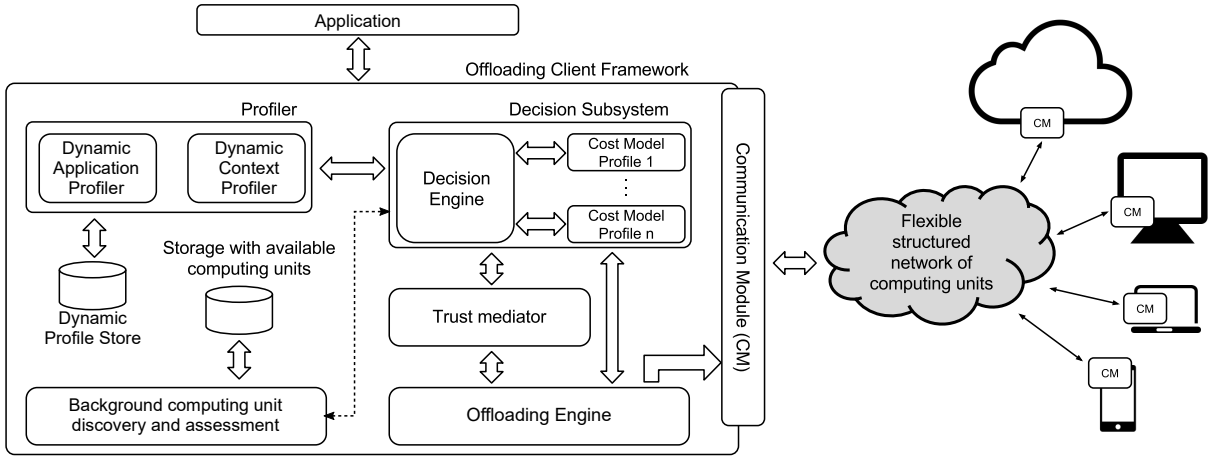


Figure 8.6: Proposed evolved and flexible client-side architecture

provides the required functionality to fulfil the security requirements *R2 - No single points-of-failure* and *R3 - Usage of multiple communication technologies* on a conceptual basis. Details about the techniques used for the integration of multiple nodes in the system and on how to discover relevant computing units are discussed in Chapter 8.3.3, a realisation of this proposal is provided as part of the implementation in Chapter 9.

This enhanced architecture aims at deployments and implementations making use of different types of computing resources in a dynamic way. Therefore, implementations will have to deal with frequent joining and leaving of computing units in the network. Computing units may qualify as legitimate at one point in time, but may not be eligible a few moments later. The architecture does not aim at including constantly available computing units only, but also those with a more dynamic availability pattern. This behaviour is supported by the flexible network infrastructure and by the background task.

8.3.2 Server Architecture

This chapter focuses on developing a server-side architecture, adhering to the high-level functional requirements from Chapter 8.2. Furthermore, the architecture development process is driven by the security related requirements from Part II. The terms *server* and *client* are only valid for a single interaction. In real world use cases, a device or computing unit may act as a client (requesting assistance) or server (providing assistance) exclusively or even in parallel. However, for developing the architecture, client and server are treated as two separate concepts. Existing frameworks only loosely define the server side. They focus on technical aspects of partitioning and offloading methods. Single computing units are likely being used by multiple clients at the same time. Without a properly defined server architecture, including security measures, client's data and processes are at risk of being exposed to other clients or untrusted administrators.

As a starting point, a straight forward server-side architecture, as illustrated in Figure 8.7 is assumed. This architecture does not map to all existing frameworks but represents their key aspects:

- Current frameworks statically bind clients to particular servers before deployment, or manually at runtime.
- In most cases, this results in computing units which are exclusively bound to one client. Therefore, no additional isolation mechanisms are applied.
- The application logic, required to run the offloaded parts, either is pre-deployed on the computing units or is attached to the offloading request.

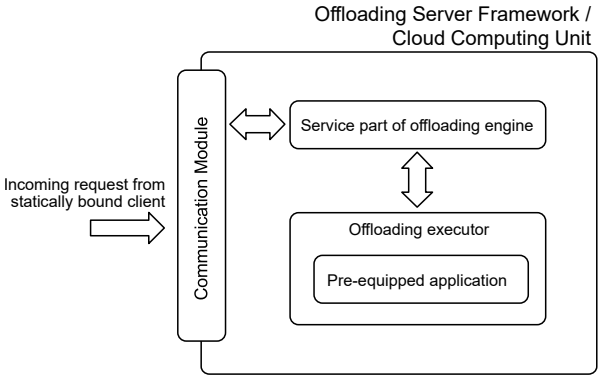


Figure 8.7: Server architecture starting point

This architecture obviously fails on all of the defined functional and security related requirements. Flexibility and lightness are hindered by the presence of a static binding to clients. Interoperability is hindered due to the exclusive focus on cloud computing units. Security requirements, as the analysis in Part II illustrates, are not considered.

To be able to apply a step-by-step methodology, similar to the approach used to develop the client architecture in Chapter 8.3.1, the functional requirements need to be transformed to the server context. The lightness requirement transfers to the organisation of available computing units and is mainly considered by the client side and network infrastructure of the system. The flexibility requirement transfers to the run-time binding capabilities of the framework to computing units. This goes hand-in-hand with a multi-user support and proper scheduling of the resources. Interoperability goes hand-in-hand with the lightness requirement. Heavyweight solutions, requiring considerable configuration, are not suited for deployment on mobile devices. In addition, interoperability clearly is an implementation aspect.

To enable the usage in multi-user environments, the framework requires a controlling instance. As illustrated in Figure 8.8, a *Hypervisor* acts as the controlling instance and keeps track of all the executed tasks. In today's environments, the term *Hypervisor* has a strong connection to virtualised environments. In fact, it is comparable to *Hypervisors* in virtualised environments regarding their duties, but the realisation is not tied to virtualisation technologies. To target the security requirement *R7 - Protection against resource over-consumption*, the *Cost Model* and *Context Profiler* are introduced in the system. The *Cost Model* component on the server-side is working analogously to the cost model component on the client-side. It enables to define usage characteristics of particular computing units. The cost model enables to define a maximum number of simultaneously executed offloading tasks, maximum utilised CPU power, maximum used memory, or maximum consumed bandwidth. Using this approach operators can configure their computing units in a way that other tasks and processes on the computing unit are not disturbed or limited in their functionality. The *Context Profiler* component supports the defined cost model by providing accurate measurements of the current status of the computing unit in terms of providing current CPU utilisation, memory usage, or bandwidth usage. Based on this collected information, the *Hypervisor* can decide if new offloading tasks can be accepted, and how much computing, memory, and bandwidth resources are available for new tasks.

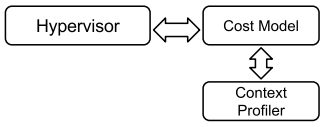


Figure 8.8: Hypervisor and cost model for the server side

Additional security enhancements are applied to target *R1 - Access Control* and *R4 - Resource's integrity assurance*. The introduced components are illustrated in Figure 8.9. The *Hypervisor* component acts as the central point of contact of the architecture. *R1 - Access Control* is targeted by introducing a *Access Control* component associated with a *Trust Store* which is under control of the computing unit operator. The *Trust Store* is not tied to a specific technology but could be realised using a certificate-based trust store. This way, the computing unit operator can control the allowed entities or provide a public available computing unit. Due to the direct relation between the *Hypervisor* component and the *Access Control* component, the *Access Control* component can even define entity-specific environments, with different capabilities. The *Trust Mediator* component targets the *R4 - Resource's integrity assurance* requirement. It acts as the counterpart to the *Trust Mediator* component in the client architecture. Depending on the used technologies and platforms a set of attestation methods is available that can be triggered by the client framework to ensure the trustworthiness of a computing unit. Concrete methods and implementations are discussed in Part IV.

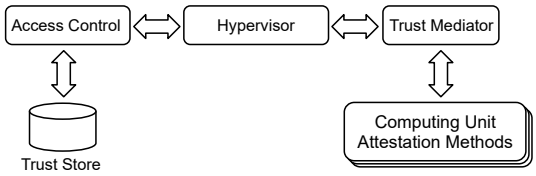


Figure 8.9: Hypervisor and cost model for the server side

The heart of the server architecture is the component where the actual execution takes place, as illustrated in Figure 8.10. At first, there are only minor differences compared to the baseline architecture in Figure 8.7 but actually it targets two requirements. Compared to the baseline architecture, the enhanced architecture provides an abstraction for the application retrieval to overcome the issue of pre-deployed applications and increase the flexibility of the system. Application retrieval is based on public or private application repositories, shown in Figure 8.11. They contain the relevant application packages globally accessible or for a pre-selected audience. Using this approach, computing units can be used for various applications without the need to do any pre-deployment. Furthermore, sandboxing of the tasks targets

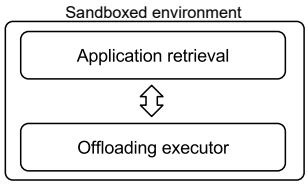


Figure 8.10: Sandboxed execution environment

security requirement *R5 - Complete task isolation*. Sandboxing prohibits that different offloaded tasks have access to each other's data. In virtual machine-based approaches, this is provided by the hypervisor, for other approaches, different measures need to be in place. Android systems, for example, make use of Linux provided measures, where each application gets assigned a unique user ID. Each file on the file system is protected by only allowing the owner of a file to read and write. Therefore, other applications, executed as another user, cannot access the data of other applications.[Faruki et al., 2015; Enck, Ongtang, and McDaniel, 2009]

Consolidating this development results in the server side architecture as illustrated in Figure 8.12. This architecture does intentionally not impose any restrictions on the utilised technologies and enables implementation for server environments as well as light implementations targeted at mobile devices. The key improvements of this architecture, compared to existing frameworks are consolidated in the following:

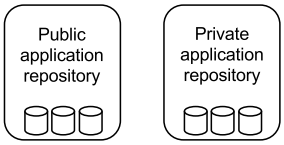


Figure 8.11: Application repositories

- Introduction of a server-side hypervisor, cost model and context profiler that keep track of the available resources.
- Using a sandboxed environment for the task executions to protect users' data.
- Introduction of an abstraction for the application retrieval to avoid pre-deployment of applications on computing units.
- Integration with the client-side by using a flexible network infrastructure.

In Chapter 9, a proof-of-concept implementation exploiting the full potentials of the architecture is provided. The implementation is equally targeted at mobile devices and server environments and still getting the best of both worlds.

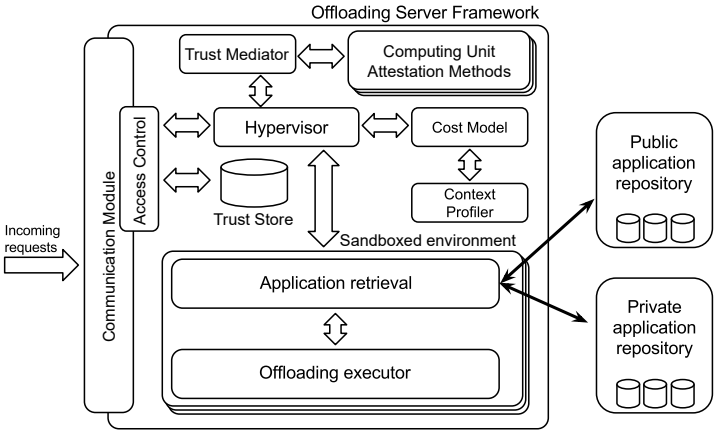


Figure 8.12: Proposed evolved and flexible server-side architecture

8.3.3 Flexible Network Infrastructure

In Chapter 8.3.1 and Chapter 8.3.2 a flexibly structured network of computing units is introduced, without going into the details of how this could be realised on the architectural and technical level. Given two nodes *A* and *B* with the intention of communicating with each other. Basically, from a high abstraction level, this intention can be reduced to two different approaches: communicating via a third party or directly communicating with each other. A direct connection between each of the devices would be reasonable but would require network configuration each time a client enters or leaves a network. The following listing compares the currently prevalent approaches:

- The first approach is the usual approach in today's Internet infrastructures. *Client-server structured* approaches (illustrated in Figure 8.13) are a simple way of connecting multiple nodes. A single server acts as the central point for all communications and processes all incoming requests. This

approach might work in a small local scope but does not scale to a global environment where large amounts of clients use a single server.

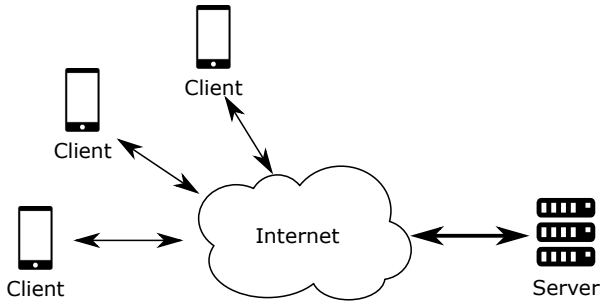


Figure 8.13: Client-Server communication

- An improved approach is a *client-cloud* approach (illustrated in Figure 8.14), where the single server is replaced by a scalable cloud infrastructure to solve the scaling issue. Load balancers are responsible to evenly distribute the incoming requests to the available units. From a high abstraction level, the *client-cloud* approach looks similar to the *client-server* approach, but *client-cloud* provides horizontal scaling capabilities out-of-the-box, hence is more powerful.

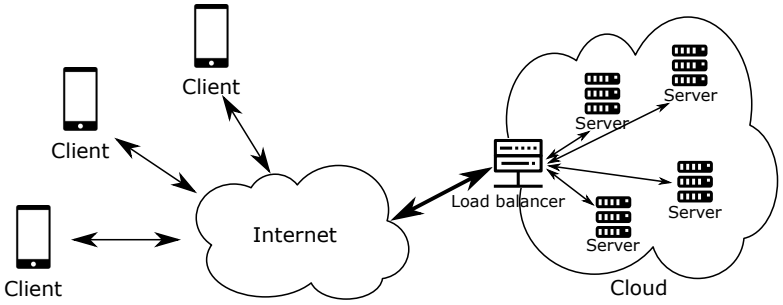


Figure 8.14: Client-Cloud communication

In theory, these approaches can also be used to interconnect and organise available computing nodes for offloading frameworks. At least the client-cloud approach is definitely capable of providing enough computing power and bandwidth. Nevertheless, outside of controlled environments, single servers and even geographically distributed cloud setups are subject to large latency variations due to unstable mobile device uplinks. Therefore, an alternative is required. Also the security analysis in Part II yields a similar result with the requirements *R2 - No single-points-of-failure* and *R3 - Usage of multiple communication technologies*. Both cannot be fulfilled with today's prevalent client-server and client-cloud approaches.

P2P networks aim at the sweet spot between centralised server/cloud infrastructures and direct communication approaches. P2P infrastructures do not require central management points and are self-organising. Still, they enable each participant to communicate with each other and often also support in building direct connections between clients (also called peers in the P2P universe). A schematic representation of P2P networks is provided in Figure 8.15. It does not represent one specific type of P2P network but represents the key aspects. Compared to the previous approaches, the underlying communication technologies are not limited to the Internet. Furthermore, clients and servers are equal entities in the network without requiring management components. This targets the security requirements *R2* and *R3* and is elaborated in Chapter 9. For the application in offloading frameworks this means that local P2P coordination points could be set up (e.g. reachable via Wifi) and only minimal transfer with the outside world is required. Therefore, P2P networks are considered as the most flexible and best approach in the field of HMCC/HMEC.

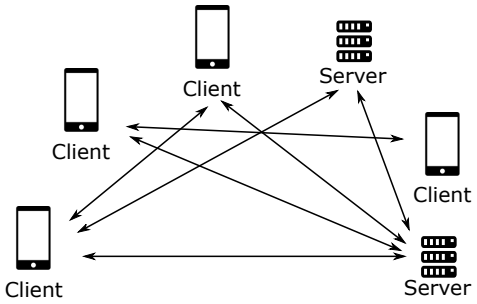


Figure 8.15: Schematic peer-to-peer communication

P2P networks are well known in the research community, and various frameworks are available. Hence, the remainder of this chapter focuses on analysing the key responsibilities of P2P networks on HMCC/HMEC systems. Low latencies to the utilised computing units is the ultimate requirement in MCC systems [Cuervo et al., 2010]. With higher latencies, the advantages of MCC approaches are eliminated. A brief introduction of the P2P network technology is provided in Appendix A to the extent as required for this thesis. In summary, the main benefit of P2P networks is that all participants in the network can be reached, regardless if direct connections to the target peers are possible. As a result, any information saved on nodes participating in the P2P network can be retrieved by any other node. In contrast to client-server-based approaches, they do not require local setup and are generally self-organising. Nevertheless, the flexibility of P2P networks does not come free of charge.

Client-server models are generally operating on TCP level to establish a connection between computers. This connection passes multiple infrastructure components (like routers) on its way to the target computer. Listing 8.1 shows the route from TU Graz' internal network to Google's well-known public DNS server. The crux of this listing is the number of hops between the origin and the target host. All intermediate hosts in Listing 8.1 are specialised and optimised for providing maximum throughput and minimal latency, still, eleven intermediate nodes are required to reach the target host. The number of hops depends on the Internet provider's infrastructure and access points to the backbone networks. A publication from the year 2000 by Begtašević and Mieghem [2000] concludes at an average hop count on the Internet of 15 to 16 hops. This, for sure, does not reflect the situation today. Further considerations in this chapter assume an average hop count of 12 as also shown in Listing 8.1.

```
 1    1 ms    <1 ms     1 ms  10.27.152.1
 2   <1 ms    <1 ms    <1 ms  192.168.0.1
 3    1 ms    <1 ms    <1 ms  129.27.200.161
 4    1 ms    <1 ms     1 ms  129.27.252.77
 5    1 ms     3 ms     1 ms  tengigabitethernet5-3.graz1.aco.net
    [193.171.21.41]
 6    4 ms     4 ms     4 ms  vlan321.wien1.aco.net [193.171.15.17]
 7    5 ms     5 ms     4 ms  bundle-ether-1-70.core1.aco.net [193.171.23.98]
 8    9 ms     6 ms     7 ms  r98-bm.cesnet.cz [195.113.179.149]
 9   16 ms    14 ms    13 ms  195.113.235.109
10   12 ms    12 ms    12 ms  r2-r93.cesnet.cz [195.113.157.70]
11   12 ms    12 ms    12 ms  209.85.251.133
12   12 ms    12 ms    12 ms  google-public-dns-a.google.com [8.8.8.8]
```

Listing 8.1: Typical route between two hosts on the Internet

P2P networks are also called overlay networks because they add another routeing layer on top of the Internet infrastructure. Participants in a P2P network could be ordinary computers or even mobile devices. They are not optimised for throughput or latency but, as an integral part of the P2P network, they may be used by other nodes to route request to the destination. Considering the Chord [Stoica et al., 2001] P2P network, where each routeing request is routed through approximately $\frac{1}{2}\log_2 N$ (N refers to the number

of participating nodes in the network) other nodes. For 10000 nodes in the P2P network, this results in about seven hops (refer to Equation 8.1) in the overlay P2P network.

$$\frac{1}{2}\log_2 10000 = 6.6 \tag{8.1}$$

Breaking this down to the IP level means that for each hop in the overlay network, 12 hops are required on IP level. Using the example from above, where a single routeing operation in the overlay network takes seven overlay-network hops, results in 84 hops on IP level, to route a single request.

With these considerations in mind, it is clear that P2P networks are not suitable to carry out the complete communication in HMEC systems. Instead of replacing low-level communication means the P2P network can act as decentralised connection service, which enables the discovery of other nodes. Furthermore, it provides assistance in establishing a direct low-latency connection (loosely adhering to the Interactive Connectivity Establishment (ICE) [Rosenberg, 2010] approach):

- A node needs to determine if it can receive direct connections or if a firewall or Network Address Translation (NAT) gateway is blocking connections. For this purpose, other nodes can try to connect.
- If no direct connection is possible, other nodes can act as gateways, similar to the role of TURN [Mahy, Matthews, and Rosenberg, 2010] servers in the ICE [Rosenberg, 2010] approach.
- As a fallback, the P2P network can still be used for the whole communication, but the offloading framework needs to consider the relatively high latency compared to direct connections.
- During the connection establishment process, the parallel P2P channel acts as a signalling channel to transmit relevant connection information like IP address, port number, or used gateways.

This analysis clearly defines the role of P2P networks in HMEC systems. They can be used as a fail-safe and distributed management entity and assist in the connection process of clients (assistance requesters) and computing units (assistance providers). All entities are equal parts of the network and can switch between requesting and providing entities.

A remaining issue is the discovery of relevant nodes. In the context of HMEC, *relevant* is defined as having a low latency in a direct connection scenario. To discover nodes with low latencies, different approaches can be adapted.

The most basic approach is to discover nodes randomly and assess their connectivity properties during the connection establishment procedure. This approach works in a network where the bigger part is composed of high performance and dedicated computing units with high bandwidth and low latency connections. In a network with many mobile devices or devices with bad connections, this approach generates a much higher overhead. Another approach is proposed by Y. Lee et al. [2008] and is based on the observation that geographically proximate nodes tend to have lower latencies. Nevertheless, only the geographic proximity of two nodes does not guarantee a low latency, but is suited as a filter on a list of nodes. An improved approach is introduced by Agarwal and Lorch [2009] called *Htrae* as a hybrid between the geolocation-based approach and a network coordinate system. The estimated RTT determined using the geolocation-based approach is used as a starting point and is mapped to coordinates in a virtual space. The distance between two coordinates is the estimation for their RTT. The coordinates are then moved in the virtual space based on actual observations to replicate the measured RTT values.

These are two possible baseline approaches to find relevant nodes without any requirements on the deployment structure. This work introduces a lightweight and more sophisticated approach, targeted at the mobile device landscape, but requiring certain deployment characteristics. The approach is introduced and illustrated in Chapter 9.2.5.2.

8.4 Chapter Conclusions

In this chapter, an improved HMEC architecture, considering the client- and the server-side, as well as approaches on interconnecting participating computing units, was developed. The methodology followed was to get an abstract view on existing frameworks and architectures and develop strategies using novel technologies to fill the identified gaps. The starting point for the development of these architectures is the analysis of existing frameworks on the one hand, and the defined security requirements on the other hand. Step-by-step, the requirements are considered by enhancing and extending the baseline architectures. Table 8.2 shows that this approach enabled to achieve a broad requirements coverage, already on the architectural level. It shows the major architectural enhancements on top and the defined high-level functional and security requirements on the left. Grey fields indicate that the component on top contributes to fulfilling the requirement on the left. Of course, each of the enhanced components on architectural level requires a corresponding implementation.

As an example, the interoperability requirement mainly is an implementation aspect. Still, the dynamic profiler contributes to the fulfilment of the requirement by adapting the analysis and decision on the offloading operation to the current system. Furthermore, application repositories enable to keep a small footprint for server environments by retrieving applications as required and also enable deployment on mobile devices. The flexible network infrastructure enables to interconnect mobile devices and dedicated servers in local and wide area scenarios. Hence, it fosters the interoperability between different architectures, devices types, and operating systems. Still, interoperability on implementation level is key.

		Client architecture				Server architecture					
		Dynamic profiler	Multiple cost models	Trust mediator	Background discovery	Hypervisor/ cost model	Access control	Trust mediator	Sandboxed task execution	Application repositories	Network infrastructure
High-level functional requirements	Lightness	■			■					■	■
	Flexibility	■	■		■	■			■	■	■
	Interoperability	■								■	■
Security requirements	R1 - Access Control			■			■				
	R2 - No single points-of-failure			■	■						■
	R3 - Usage of multiple communication technologies				■						■
	R4 - Resource's integrity assurance on technical level			■				■			
	R5 - Complete task isolation								■		
	R6 - Non-repudiation			■			■				
	R7 - Protection against resource over-consumption					■					

Table 8.2: Requirements coverage on architectural level

The result makes use of existing building blocks like the profiler and decision engine, refines and redefines these building blocks and also inherently improves the definition of complete framework components like the server-side components. The developed architectures are a clear enhancement beyond the state-of-the-art. To a large extent, they close the gaps related to lightness, flexibility, interoperability, and security-awareness on the architectural level. This taps new fields of application for HMEC solutions. This architecture acts as the foundation for all further considerations regarding the implementation and evaluation, but also when it comes to detailed security extensions in Part IV of this thesis.

Chapter 9

Implementation

The weak spot of existing solutions, from an implementation's point-of-view, is their focus on the framework's client side and only considering selected topics. The implementation, based on the architecture defined in Chapter 8, scrutinises on both the client-side, based on Chapter 8.3.1 and the server-side, based on Chapter 8.3.2, with the goal to analyse and evaluate the approaches from both perspectives.

The requirements on architectural level, summarised as *lightweight*, *interoperable*, and *flexible*, have clear mappings to targets on implementation level:

- A *lightweight* offloading system is reached by organising the available computing units in a decentralised manner, without relying on single computing units. This goes hand-in-hand with the fine-grained system to specify which parts of an application should be considered for application offloading.
- *Flexibility* is achieved by enabling a run-time binding to available computing units, also considering current device and connection conditions.
- *Interoperability* is assured by not focussing on single programming environments for particular ecosystems, but by relying on programming languages which are equally compatible with all major environments.
- Use cases, where processing of sensitive data is involved, require additional measures. They can be implemented by providing explicit control of the devices used for offloading, as well as explicit control of the data flows in an offloading scenario. Security is discussed in detail in Part IV of this thesis.

Other key aspects of MCC, like the general offloading approach or task scheduling are already well established and proven in existing frameworks. To achieve a broad distribution and acceptance of the developed offloading framework, interoperability is key. Interoperability enables to use e.g. computing units running on iOS to accept offloading requests from Android devices. In general, this is a major obstacle of other frameworks. The goal is to support all common mobile- and desktop-devices, with a single implementation.

Currently, applications developed using web-technologies are the only applications with a broad support on various devices even running on different hardware architectures. Creating an offloading-framework based on web-technologies enables offloading for online web-applications, but also offline applications created using cross-platform frameworks like Apache Cordova[1].

A new class of programming languages uses JavaScript as intermediate language. Programs are developed in a higher-level programming language, and are compiled to efficient JavaScript source-code.

[1] https://cordova.apache.org/

This approach is e.g. used by Microsoft supported TypeScript[2] or Google supported Dart[3] programming languages. TypeScript does not define a complete new syntax. It is based on the ES-6[4] recommendations and provides a compiler to ES-3, ES-5 and ES-6. Google in contrast, with their Dart approach, defines a whole new syntax and provides a Dart-to-JavaScript compiler to create ES-5 compatible JavaScript source code. The advantage of Dart is that recently the use cases of Dart were extended beyond web-based applications and native command-line environment support. Compiled to JavaScript, Dart code can be used to create Apache Cordova applications, but additionally, with Flutter[5], native-Dart can be used to create high-performance mobile applications targeting iOS and Android devices. This makes the Dart programming language very attractive to create an offloading framework.

9.1 Operation Environment

This chapter settles the scene for the offloading framework by defining the boundaries of the environment. It is not feasible to create frameworks which cover the whole landscape of available applications and devices and still sticking to properties like interoperability with a single implementation. Therefore, an implementation requires clear constraints on the environment where it should operate and which applications are targeted. In the previous chapter it was elaborated on the widespread use of web- and related technologies. In this chapter the environment is specified in detail, together with concrete types of applications.

Due to the widespread use cases, this offloading framework will be the first targeting the Dart programming language. Therefore, it can be used in classical web-environments and in Dart specific use cases:

- *Web-application created using Dart*: For web-applications, Dart is an alternative to JavaScript, and still can fully interoperate with JavaScript libraries. The framework will not be capable of offloading calls to JavaScript code. To maintain interoperability between devices and different environments, only Dart code will be supported.
- *Cross-platform applications created using Dart*: Cross-platform applications, created using web-technologies, are executed in web-views on mobile devices. Web-views are browser instances, embedded in a native application. Therefore, they are equally capable of using the full available feature-set. Plugins enable access to native platform features.
- *Native applications running in the Dart virtual-machine*: The Dart VM is an execution environment where native Dart applications can be executed without converting the applications to JavaScript.
- *Cross-platform applications using Flutter*: Flutter can be used to create Dart-based cross-platform mobile applications without converting to native applications and without involving web views, by running a high-performance Dart-VM on each mobile device.

To enable the offloading framework in applications, changes on the source-code level are required as described in Chapter 9.2. Therefore, the offloading improvements cannot be applied to unchanged applications. It may be possible to create add-ons which transparently apply the framework to e.g. existing web-applications but this is not the primary goal of this thesis and is considered as future work.

Beside the operation environment, available technologies are an important aspect. Technologies enable the realisation of properties like flexible communication and interoperability. Therefore, a solid set of baseline technologies is elaborated in the following:

[2] https://www.typescriptlang.org/

[3] https://www.dartlang.org

[4] http://www.ecma-international.org/ecma-262/6.0/

[5] https://flutter.io/

- For web-environments, support for ECMAScript-5 is required. For native Dart environments, support for core dart libraries is required.
- To enable a flexible communication, without being tied to strict client-server models, enabling technologies are required. Clearly, Web Real-Time Communication (WebRTC)[6] is the technology of choice if available. Still, WebRTC has not reached a broad availability beyond web-browser environments yet. As an alternative, support for WebSockets [Fette and Melnikov, 2011] is required.
- Collaboration between different nodes always involves cryptographic operations. Therefore, a hardware-supported implementation of cryptographic primitives should be available. In web-browser environments this is provided with the support for the WebCryptoAPI[7], in native Dart environments wrappers to native device implementation should exist.

Based on these technologies a sustainable framework is developed and evaluated in the following chapters, which adheres to the defined constraints and works in the boundaries of the operation environment.

9.2 Offloading Framework

This chapter elaborates on the developed Dart-based offloading framework. The framework is based on the architectures developed in Chapter 8. On the architectural level, a separation was introduced between the client- and server-architectures. This is a logical step and fosters the understanding of the unique components of the framework. Nevertheless, the concept of HMEC does not foresee a clear separation between client and server components. Therefore, as a first step the client and server separation

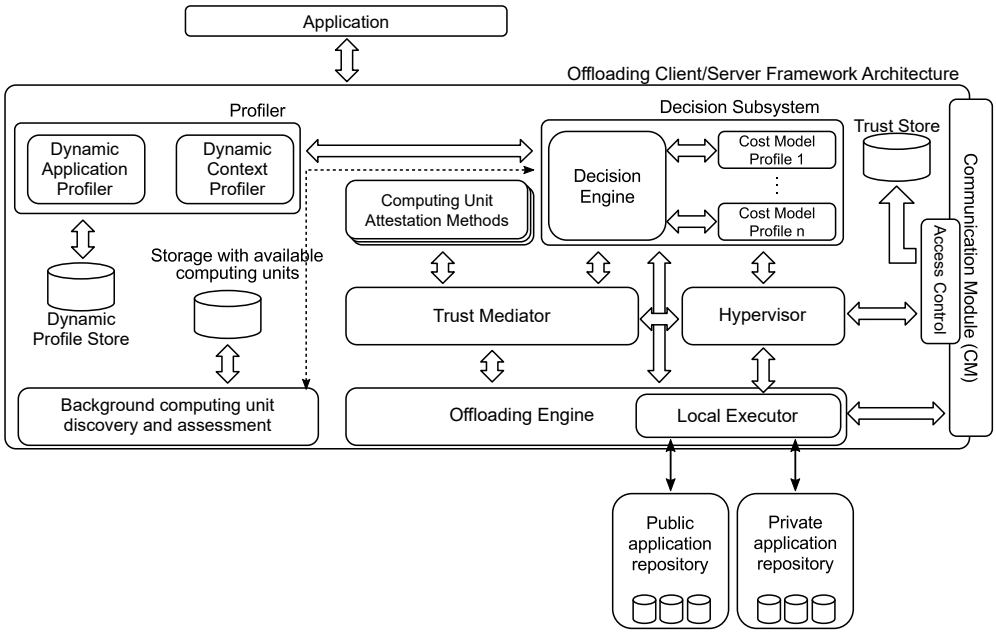


Figure 9.1: Consolidated client and server architecture

is consolidated into a single architectural representation shown in Figure 9.1. The result is an architecture which is already closer to the actual implementation architecture, where components occurring on the client and on the server side can be reused. Compared to the separated architectural view, the

[6] https://tools.ietf.org/html/rfc6455
[7] http://www.w3.org/TR/WebCryptoAPI/

complexity is substantially increased. Still, this is a necessary step for the development of the implementation architecture and to achieve the required levels of interoperability and flexibility. Hence, the implementation-focused architecture and the implementation also do not preserve this strict client-server separation. Concrete links to the consolidated architecture are established. The following chapters elaborate on the translation of these concepts to an actual HMEC framework implementation by starting at an implementation-focused architecture, which provides a technology-focused view on the consolidated architecture, and then elaborating on the different implementation aspects of the HMEC framework.

9.2.1 Implementation-focused Architecture

For the development of the implementation-focused architecture, the operation environment from Chapter 9.1 was considered, as well as the fact that interconnecting nodes using P2P networks is the most efficient and flexible way. On architectural level, the focus lies on identifying the different components and determining the relation between the components. The implementation-focused architecture already considers suitable technologies and provides a more detailed view on the concrete implementation. This development process also includes the consolidation of components, if possible, to maintain simple component surfaces. The five components, identified on architectural level, act as a starting point: *Offloading Engine*, *Trust Mediator*, *Decision Subsystem*, *Communication Module*, and *Profiler*. They have distinct responsibilities with many connections to other components.

The *Hypervisor* has a strong relation to the *Offloading Engine*. It actually acts as the central connection point for incoming offloading requests. The *Hypervisor* has equal responsibilities compared to the *Offloading Engine*. Therefore, it is integrated with the *Offloading Engine* component. The *Computing Unit Attestation Methods* enhance the *Trust Mediator* with methods to determine the trust level of local or remote computing units. Clearly, they have a direct relation to the *Trust Mediator* component and are integrated without impacts on the external dependencies of the *Trust Mediator*. The *Access Control* component has a one-to-one relationship with the *Trust Store*. Actually the *Trust Mediator* may also contain a *Trust Store* component as part of one of the attestation methods. Therefore, the *Access Control* component is also integrated into the *Trust Mediator*. The node discovery task is integrated with the *Communication Module* because it already hosts all communication related sub-components.

Figure 9.2 illustrates the technological-enhanced architecture. The six distinct blocks can briefly be described as the following:

- The *Offloading Engine* component acts as the coordination point in the framework and performs the actual technical offloading process, as well as accepts incoming offloading requests from requesting nodes.
- The *Profiler* component collects application and device profile values and provides these values to other components.
- The *Decision Subsystem* evaluates if a particular task should be offloaded or executed locally based on the provided profiling values, as well as connected P2P performance values.
- The *Communication Module* discovers and manages computing units for the framework and provides a communication link to other computing units. Furthermore, the *Communication Module* gathers performance values of the connected P2P nodes.
- The *Security Subsystem* enables on the one hand to establish trusted end-to-end connection from one computing unit to another and is capable of assessing the trustworthiness of the local and remote computing nodes. This enables the framework to selectively offload to trusted computing nodes.
- The *Dart-based Application* embeds the framework using means as described in the remainder of this chapter.

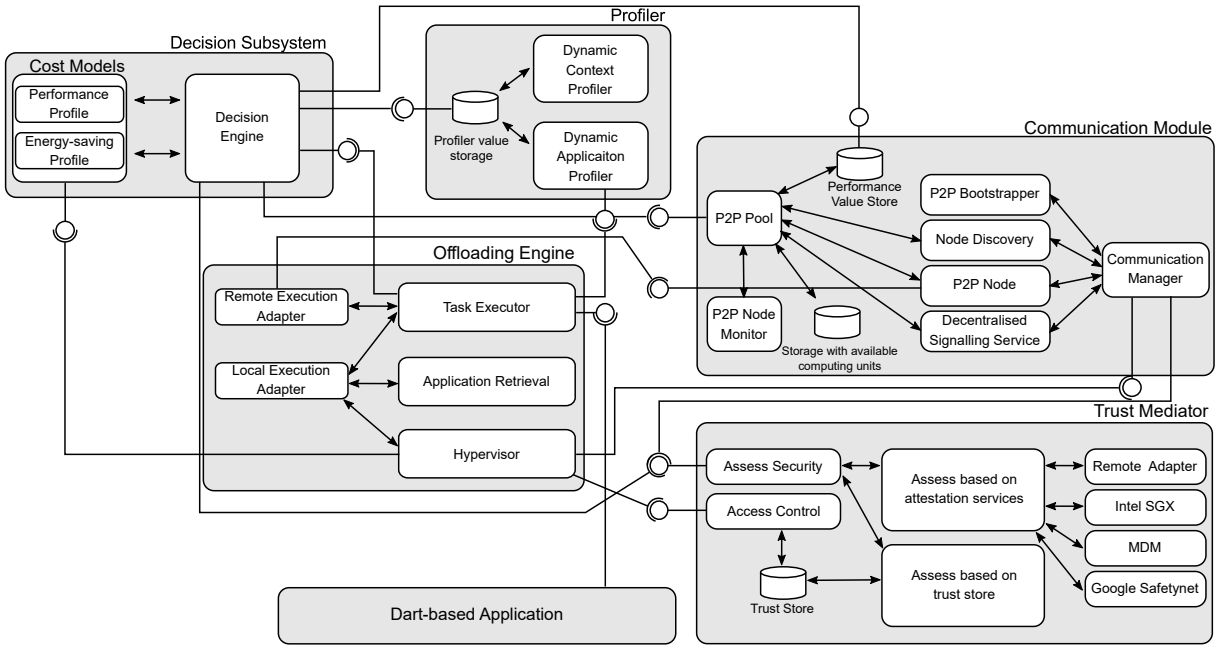


Figure 9.2: Implementation architecture

The heart of the framework is the *Offloading Engine*. It has two modes of operation: (a) applications use the *Offloading Engine* to offload remotely executable tasks, and (b) the *Hypervisor* listens for incoming offloading requests issued by other nodes. Both cases use the same components, but the *Hypervisor* skips the decision process for incoming requests. The *Hypervisor* mode also includes sandboxing of the offloaded tasks. Details of different sandboxing approaches are discussed in Chapter 9.2.6. The *Offloading Engine* intercepts calls to remotely executable tasks of the application and decides if a particular task should be executed locally or remotely. For local executions the offloading engine invokes the *Local Execution Adapter*, which executes the task locally and returns the calculated result. For remote executions, the task is delegated to the *Remote Execution Adapter* which migrates the task to one of the available computing units, with respect to the security requirements of the particular task.

The *Trust Mediator* and the introduced means to control the security requirements of particular tasks are introduced in Part IV of this thesis in detail. Overall, the *Trust Mediator* component offers trust store based access control for the *Hypervisor* component and attestation mechanism integration for the local device, as well as an adapter to assess remote computing units.

The *Communication Module* is the central communication interface to the P2P network. It manages the available and discovered computing units and is consisting of multiple P2P related sub-components. The *P2P Pool* always keeps connections to a configurable amount of computing units. These computing units are selected based on a calculated score of their current performance values. Furthermore, the *P2P Node Monitor* keeps track of the available computing units and constantly measures their availability and performance values. The *P2P Node* is an abstraction of the peer's communication interface and is used for seamless communication throughout the framework. If a node is not available anymore, or does not meet the performance requirements any longer, it is dropped and a new node is acquired. The discovery and constant monitoring of available computing units is performed by a separate background task. This task is automatically scheduled during the instantiation of the framework. The overall goal of this approach is to always have a list of high-performance computing units waiting for offloading tasks. The activity diagram of the background discovery and assessment task is illustrated in Figure 9.3 and follows the below procedure:

1. Check if all properties are met (size of computing unit repository, maximum latency). If properties are not met, continue with Step 2.

2. Discover new computing units using the P2P network with signalling-server assistance.

3. Try to establish a direct connection to the discovered computing unit. If a direct connection is not possible, continue with Step 2.

4. Determine the current latency to the computing unit and queue it into the resource repository.

5. Request an end-to-end-secured connection to assess the trust level of the computing unit if requested.

In addition, the framework continuously gathers latency values from connected peers in order to react to slow resources and changing network conditions. The framework aims at resource-constrained mobile devices with potentially unstable network connections. In these environments it is observed that latency can increase rapidly for a single measurement, and be back at normal level for the next measurement. Therefore, the last few collected latency measurements are kept and the median value is used. This way, single outliers do not trigger expensive discovery and direct connection procedures.

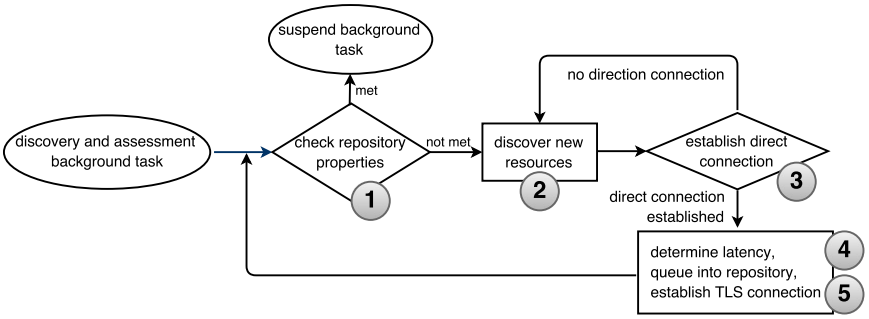


Figure 9.3: Resource discovery and assessment background task

The developed P2P network approach is not meant to be used for the whole communication, but to discover suitable computing nodes and establish a direct connection to minimise latency. For this purpose the *Communication Module* offers multiple components. The *Communication Manager* coordinates the associated P2P relevant sub-components. The *P2P Bootstrapper* sets-up and joins the network. The *Node Discovery* sub-component enables the discovery of nearby and suitable computing nodes, participating in the network. The *Decentralised Signalling Service* sub-component enables to establish secure direct connections to other, discovered computing nodes. To establish a trust relationship between the connected computing nodes, the *Trust Mediator* provides measures to assess the trustworthiness of remote nodes.

The *Profiler* component constantly profiles the devices' context values and observes the application execution of local and offloaded tasks. It saves the gathered values to the *Profiler Value Storage* which again refines the decision process. The *Decision Subsystem* considers the defined cost models and evaluates the collected profiler values, as well as performance values of discovered remote nodes. Based on these values it decides if the execution should take place locally or remotely and proposes a computing unit. The cost models are also relevant for the *Hypervisor* mode to control the amount of locally consumed resources.

At the bottom of Figure 9.2, the applications is plugged-in. This could either be an application making use of the offloading framework, and occasionally requesting assistance from other computing units, or could be a server application, which initialises the offloading framework in *Hypervisor* mode. Basically, an application can also make use of both execution modes, depending on the cost models used by the application. Acting as both, client and server, only makes sense in feature-based offloading scenarios, where single devices offer features not available on other devices.

9.2.2 POWER - A Dart-based Offloading Framework

POWER [Reiter and Zefferer, 2015b] is developed to validate and prove the feasibility of the architectures on implementation level. It is based on the Dart programming language. Dart makes use of several concepts, which can be re-used to enhance the implementation and solve common problems of HMEC and MCC frameworks.

- *Pub package manager:* The pub package manager, is a repository used for managing dependencies of applications during development. For deployment, packages are collected into a single, application-specific repository. In the HMEC use case the challenge persists to get the source of tasks (and its dependencies) to the selected remote computing units. Basically two options were examined: (a) one option is to transfer the source of tasks in their offloading request, with the risk of impacting performance substantially for large tasks, and (b) another option is to use a pub package manager like mechanism, where only identifiers are transferred and the computing unit downloads the source from public or private repositories. The mechanisms are detailed in Chapter 9.2.4.
- *Transformers:* Transformers are small Dart applications attached to the build process. They offer built-in methods for e.g. source code pre-processing. Dart transformers are used to enable the developer to only loosely couple the application to the offloading framework, by applying annotations to the source code at development time. At compile time, the annotations are processed by the transformer and coupled with the concrete offloading framework. The transformer approach is detailed in Chapter 9.2.3.

The offloading framework uses a plug-in approach to easily enhance applications with computational resources from other computing units. The overall structure of the offloading framework follows the architecture as defined in Figure 9.2. Dart is focused at an asynchronous programming model, synchronous operations are also supported but developers are strongly encouraged to use the asynchronous model. Using this programming model, a method returns immediately without the actual computation result, but with a promise to provide the value in the future. Once the result is available the application is notified. In the meantime, the application can continue and probably schedule more tasks, or wait for specific promises to provide their values.

The flow of processing tasks in Dart is illustrated in Figure 9.4. Two queues where tasks are scheduled exist, whereas the microtask queue is used for prioritised tasks and the event queue is used for ordinary tasks, scheduled by the user or events from the environment. With the instantiation of a new asynchronous promise return value, and an associated computation, a new entry is added to the event queue and is scheduled for later execution by the Dart framework. This processing model is predestined for an integration with offloading frameworks because the event queue already contains parts of the application in form of tasks.

To achieve this level of integration, the framework uses the method-based offloading approach, but due to its modularity it can easily be extended to also support a class-based approach. POWER uses the Dart-provided annotation system to mark tasks of applications which should be considered for offloading by the framework. In the simplest case, an annotation is applied without providing any attributes as shown in Listing 9.1. Due to the asynchronous programming model, it is irrelevant for the application flow where a computation takes place. The result just needs to be available at the created asynchronous promise once the computation finished.

```
@OffloadMe()
Future method(){
<code>
}
```

Listing 9.1: Applying the marker attribute to a method

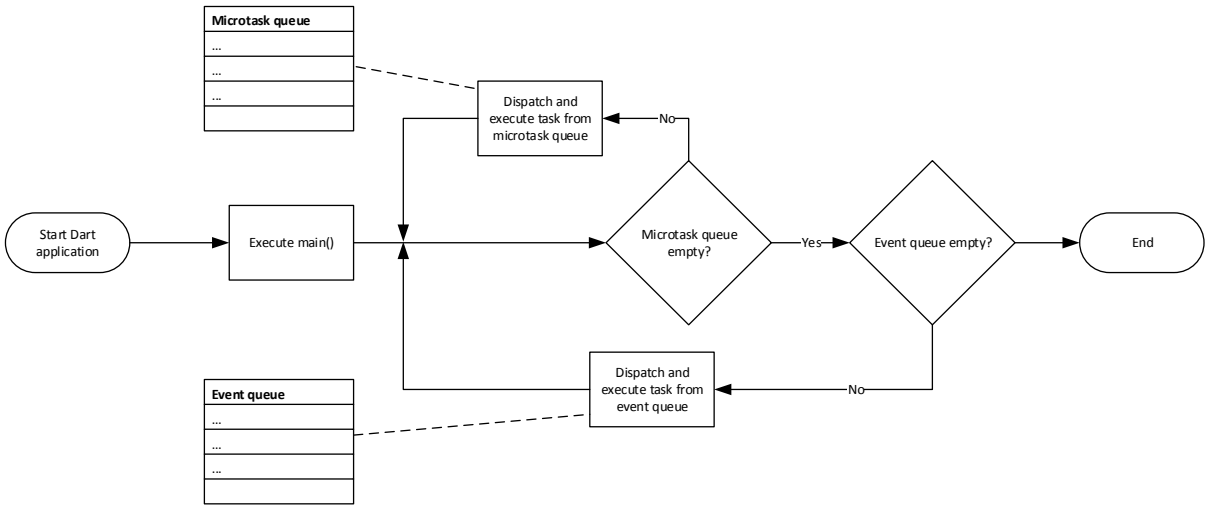
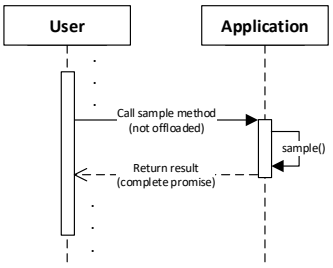


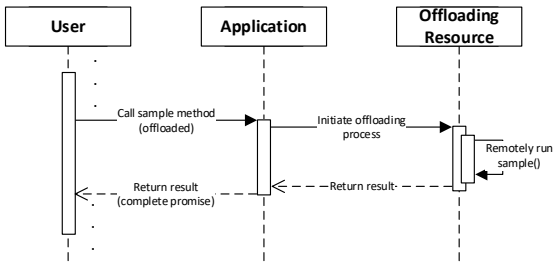
Figure 9.4: Dart event loop

Only applying annotations to tasks has no influence on the processing of the application or the application flow yet. Transformations of the source code are required to automatically invoke and transform control to the offloading engine. The concrete implementation of these transformations are subject of Chapter 9.2.3. For the rest of this chapter it is assumed that the offloading engine is invoked and control is transformed to the POWER framework if annotated methods are invoked.

The offloading engine, as the application-side entrypoint to the framework, offers interfaces which takes a reference to the offloading candiate in terms of application ID, method ID and a function/method-pointer, to directly call a method locally. An unmodified local and a modified remote execution flow is shown in Figure 9.5a and Figure 9.5b. An asynchronous method is called which performs some kind of computation. With the completion of the calculation, also the issued promise resolves to a concrete value. In a local execution scenario the flow is clear and straight forward. In an offloaded scenario, the original call to the calculation is interrupted, the calculation is migrated to another computing unit and finally the original calculation routine is executed there. After completion the results are passed to the initiating computing node, and are reintegrated into the application.



(a) Unmodified execution flow



(b) Modified execution flow induced by the transformer

Figure 9.5: Illustration of unmodified and modified execution flows

A more detailed view of the components involved in the offloading process is shown in Figure 9.6. The offloading engine is called by the application with the request to execute a specific task. First the offloading engine decides, with the help of the decision engine, if offloading is beneficial. Cuervo et al. [2010] estimate the execution time of a particular task, by basing their decision on past execution times of the same task. They do not consider the impact on the execution time of different input values. Still,

they found the approach sufficient and with virtually no impact on the overall performance for mobile applications. This approach is enhanced by also considering the provided input data. If a task is called more than once with the same input data, a more accurate estimation of the execution time can be made. If no profiling information is available at all, the decision engine requests and measures a local execution, to get a solid baseline. With profiling information available, the framework selects one of the available computing units and starts with the offloading operation. Furthermore, the decision engine is prepared to evaluate security requirements of tasks, and matches the security requirements with trust levels of computing units as introduced in Part IV of this thesis.

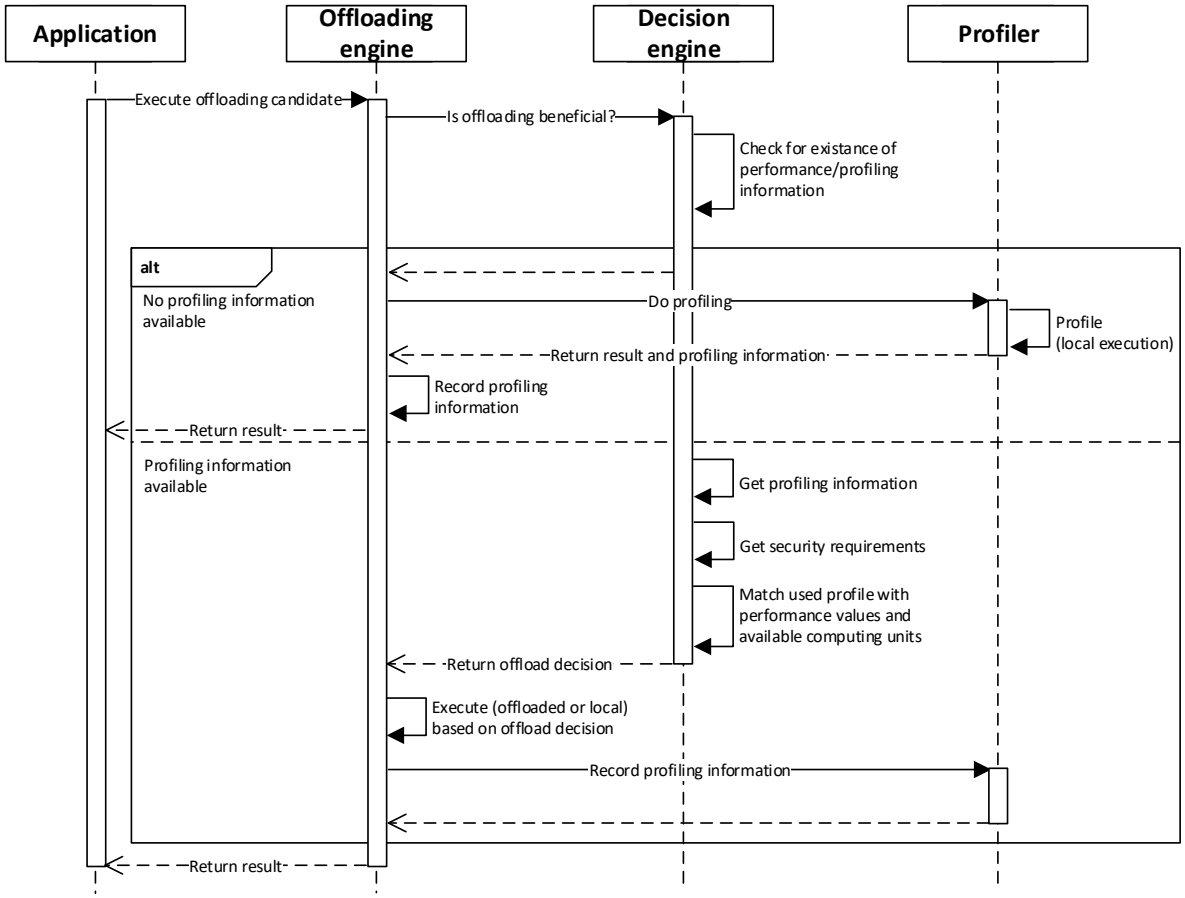


Figure 9.6: Sequence diagram of the offloading process

For offloaded executions so-called "*execution-profiles*" are introduced: the high-performance profile and energy-saving profile. Both influence the offloading decision process. Using the high-performance profile, the performance of the device is maximised by basing the offloading decision on Equation 9.1, containing the following values:

- T_{local} denotes the local execution time coming from the profiling information.
- $T_{latency}$ is the latency to the selected computing unit.
- $T_{preparation}$ is the predicted time, required to transfer the necessary state and input parameters to the computing unit. The estimation is based on continuously recorded performance values of the computing unit.
- T_{remote} denotes the predicted execution time on the remote computing unit. This value is based on historic execution times. It can only be roughly estimated, but tests of Cuervo et al. [2010] show

that most methods have similar execution patterns, independent on the concrete provided values, at least in current mobile applications.

$$T_{local} > T_{latency} + T_{preparation} + T_{remote} \tag{9.1}$$

If multiple computing units are available, the equation is evaluated for all available computing units, to minimise the execution time.

The energy-saving profile, in contrast, mimics the performance of the local device and maximises battery lifetime. Offloading computational intensive tasks still consumes energy, but as seen in Chapter 9.3, offloading and maintaining the performance of the device causes high savings in energy consumption. Therefore, the offloading decision is based on Equation 9.2. The evaluation and selection of available computing units is similar to the evaluation process of the performance profile. During the evaluation, one additional parameter T_{wait} is involved. It is used to block the executed task, and maintain the performance of the device. For the performance profile $T_{wait} = 0$, the execution would result in a better performance, but also in a higher energy consumption compared to the energy-saving profile.

$$T_{local} = T_{latency} + T_{preparation} + T_{remote} + T_{wait} \tag{9.2}$$

9.2.3 Compile-Time Transformer

Following the previous chapters, there is no tight connection between the application and the offloading framework. The applied annotations have no impact on the runtime of the application, and can still be executed without applying the offloading approach. The only changes the application developer needs to apply are annotations to tasks which should be considered for offloading.

The Dart environment offers a concept called *transformers*. They pre-process all the source code files of an application. A typical transformer for web-applications is, for example, to produce a minified version of a JavaScript library or to combine all the used CSS files into a single file. To integrate the POWER framework into applications by just applying annotations, a transformer is developed to tightly couple the concrete implementation with the targeted application. The transformer analyses the source code and processes all annotated parts of the application. The output of the transformer is twofold. The original source code is modified to get an application flow as described in Figure 9.5b and Figure 9.6, furthermore a unique ID is assigned to unambiguously communicate the offloaded task to remote computing units. This enables the offloading engine to take control of the application flow and migrate the execution to another computing node if required. This only relates to the client side of the operation. For the server side of an offloading operation, the source code of the particular tasks, identified by the unique ID as assigned during compilation, is required for a successful execution. Moreover, the server side can be executed on dedicated Dart computing units, but also in web environments. Therefore, the tasks are required as Dart source code and converted to JavaScript source code. All these files are prepared by the compile-time transformer.

Technically, the transformer performs the following transformations, considering a simple example as shown in Listing 9.1. Applying the transformer on the application creates a statically available instance of the offloading framework *engine*, which is initialised at program start-up. It discovers and keeps track of available computing units automatically. The result of the transformation for one single method is shown in Listing 9.2. The original method is encapsulated and an *execute* call involving the offloading engine is added. The *execute* call contains a pointer to the original method to enable an efficient local execution, arguments, the assigned unique method ID, the original path to the source file, a unique package ID also generated during compilation and the package name of the Dart package. Furthermore, the engine accepts the hash of the source code for the considered part. This way, the server can verify that the retrieved code is unchanged. With this information in hand, the remote offloading engine can uniquely acquire and pre-load the sources for a specific task and calculate its result on request.

```
Future myMethod(){
var m=() { <your code here> };
return engine.execute(m, [], "sample", "example/example.dart",
  "samplePackage-0.0.1", "samplePackage", SHA256(<Dart source code>),
  SHA256(<JavaScript source code>));
}
```

Listing 9.2: Result after parsing and processing the marker attribute

9.2.4 Repositories

The Dart programming language and its *Pub*[8] application is designed around the paradigm that all the packages can be retrieved from a central repository. To publish new packages, or new versions of an already existing package, the uploader needs to authenticate using its Google account. The uploader of the initial version, has control over the particular project, and decides who is allowed to upload new versions of the package. These approaches are enhanced to support HMEC scenarios.

Each application and each unique task of an application considered for offloading gets assigned a unique ID by the compile-time transformer. The source of each task can be downloaded from a central repository. Although central does not necessary mean that all operated servers need to use the same repository. During the actual offloading operation, the server downloads the corresponding files from the repository and compares the received SHA256 hash of the source code, with the actual SHA256 hash of the downloaded source code, applies the current state of the application as received from the client, and executes the task. Using this methods neither the clients nor the servers need to do any complex code analysis for dependency detection during run-time. All dependencies are resolved at compile time and are included in the downloaded file.

Different types of offloading servers have different requirements. Browser environments are not capable of executing Dart code and command-line environments are not capable of executing JavaScript code. Therefore, the repository offers both, the Dart application and the already transformed JavaScript application. The repository accepts connections via standard HTTP means, therefore is subject to the cross-origin limitation of web-applications. Thus, repository implementations need to apply the *Cross Origin Resource Sharing* (CORS)[9] paradigm to enable a cross origin usage of the repository.

To access the package repository it is differentiated between accessing the Dart application and the transformed JavaScript application. A mirror of the complete Dart application, including all its dependencies, is accessible using the following URL schema: `https://url-to-repository/package-repo/<package-id>/`. For JavaScript each task is served as a separate file, identified by its ID according to the following schema: `https://url-to-repository/package-repo/<package-id>/js/<offload-id>.js`.

9.2.5 Communication Technologies

Advancing the conceptual level to an implementation level introduces new constraints from the operation environment and requires to clearly define the expected key aspects of the communication technology. The communication technologies help in fulfilling the key goals of achieving a *lightweight* and *flexible* solution:

- A decentralised approach avoids single points of failure and also enables operation even without an Internet connection.
- Due to the nature of HMEC systems, computing units are frequently leaving and joining the network. A decentralised P2P approach adapts to these behaviour and scales accordingly.

[8] `https://pub.dartlang.org/`

[9] `http://www.w3.org/TR/cors/`

- Suitable nodes, with low latency and high performance values, need to be discoverable easily. Available nodes are organised in a P2P like structure which enables a decentralised discovery process.

Various technologies are available which enable a realisation given the defined operation environment from Chapter 9.1. Classical web technologies such as client-server HTTP or AJAX are ruled out due to their inflexibilities. Classic client-server HTTP follows a strict request-response terminology. Calling a web page issues multiple HTTP requests, the server afterwards has no way of contacting the client. AJAX is an improvement in terms of flexibility and enables applications running in the browser to dynamically issue further requests. This way, web applications can simulate a server to client communication by constantly polling the server for updates. Furthermore, AJAX is constrained by the same-origin-policy[10] that only allows AJAX requests to the same scheme, host and port combination as the web application was requested from. Although workarounds are available [Saiedian and Broyle, 2011] to circumvent the same-origin-policy, they are still constrained by only having client-server connectivity and no direct client to client connectivity at all.

Beside these classical web technologies, new web technologies like WebSockets [Fette and Melnikov, 2011] and WebRTC[11] are available which fit into the defined constraints and provide the required capabilities. Based on these technologies, the following chapters introduce two distinct approaches on how to build a communication layer, suitable for HMEC use cases and respecting the defined operation environment.

9.2.5.1 Decentralized Peer-to-Peer Framework in Web Environments

The first approach is completely based on the utilisation of WebRTC. WebRTC, as illustrated in Figure 9.7, is a new web technology which allows two clients, for example browsers, to directly communicate with each other without requiring an intermediate server. It uses technologies to circumvent NAT gateways with the support of Session Traversal Utilities for NAT (STUN) server and also provides connectivity if no direct connection is possible by using Traversal Using Relay NAT (TURN) servers. The limitation of WebRTC is that a separate signalling channel is required to exchange initial connection informations. This signalling channel can for example be realised using AJAX to one or multiple coordination servers. WebRTC is already used in various scenarios: Swarmify[12] uses WebRTC to create a

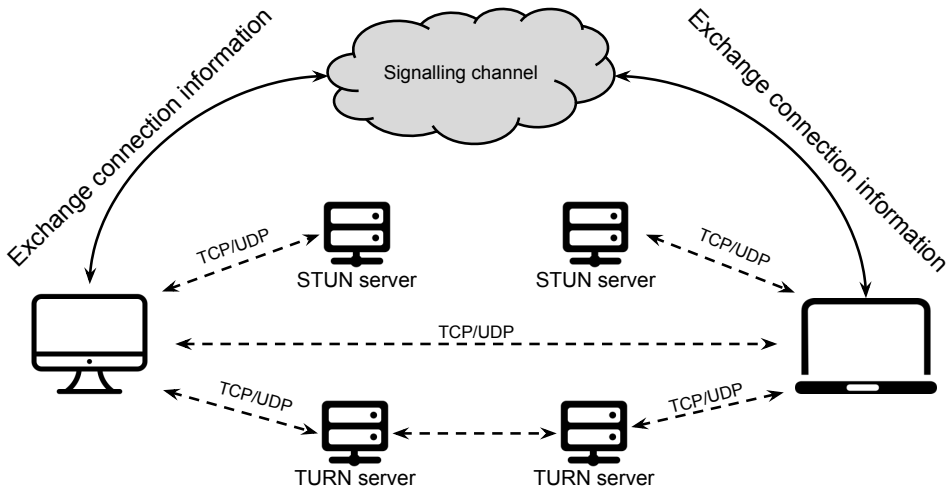


Figure 9.7: WebRTC concept [Reiter and Zefferer, 2016]

[10] https://www.w3.org/Security/wiki/Same_Origin_Policy
[11] https://www.w3.org/TR/webrtc/

distributed client-powered content delivery network, the recent Linux version of Skype[13] uses the audio and video capabilities of WebRTC, and the system developed by Burgstaller et al. [2016] uses WebRTC as an anonymous communication layer.

In this chapter, WebRTC is used to provide the foundation to build and extend a P2P framework with capabilities as required by HMEC systems. The implementation is based on the work of Dias [2015]. He introduces a WebRTC-based distributed hash table (DHT). It is the first implementation using pure web technologies. Using this approach, all nodes (e.g. browsers) can communicate with each other, without requiring a direct connection, and without requiring any server infrastructure once the connection is established. It uses WebRTC as its direct node to node transport mechanism, nevertheless it requires a central rendezvous point to exchange connection information for their initial connection. Due to limitations the central point cannot be omitted.

In the WebRTC-explorer system, as proposed by Dias [2015], each node has several up- and downlink connections to known nodes. Each node is assigned a random 48bit ID and the connections to other nodes are arranged according to this ID. To route a packet through the network, a node sends it to a known node with the closest ID. If the node exists and is integrated in the network, it will receive the packet on the shortest possible path.

This early implementation approach was advanced in different directions. The rendezvous point is extended to also act as a resource manager. All nodes register at the rendezvous point with the ability to attach attributes to the registration request. Nodes which require additional computational power, query the resource manager and receive a list of potential resource providers. To find the best performing computing unit in terms of latency, a geo-location based approach is foreseen, where nodes close to the requester's geographic location are returned. This already results in low RTT values [Y. Lee et al., 2008] in many cases. A further extension applied is the support for transparent end-to-end security. Using the forge[14] library the TLS protocol is invoked. Trust is established through a local trust store, where end user certificates are stored. The trust store is provided as part of the application. A full-fledged certificate validation and revocation checking is not possible from within the browser environment, without support from external systems.

The extended WebRTC-explorer can be considered as overlay network and furthermore was extended to also act as a decentralised rendezvous point. This way, once a connection to the overlay network is established, no central servers are required to directly connect two nodes. Still, the end-to-end security transparently operates on overlay network connections and on direct connections. Using this approach, nodes enter the overlay network via a centralised rendezvous infrastructure, but operate in a decentralised manner afterwards.

Due to design decisions in the WebRTC specification, it is not possible to completely remove the dependencies to central components and still only rely on the WebRTC technology. Therefore, an improved approach in terms of reliability, but with enhanced dependencies is introduced in Chapter 9.2.5.2.

9.2.5.2 Super Node-based Peer-to-Peer Framework

As shown throughout this work, P2P networks are best suited to fulfil the requirements of HMEC frameworks on the communication technology. The previous chapter highlights, that trade-offs between requirements and the realisation need to be accepted due to technical constraints. With the previous approach it was discovered that web-technologies do not allow to completely eliminate centralised components. With the approach solely based on WebRTC, a signalling channel connected to rendezvous points always needs to be available. Furthermore, integrating mobile devices into the coordination P2P network is dicey due to their unreliable availability and their deviations in latency.

[12] https://swarmify.com/

[13] https://community.skype.com/t5/Linux/Skype-for-Linux-Alpha-and-calling-on-Chrome-amp-Chromebooks/td-p/4434299

[14] https://github.com/digitalbazaar/forge

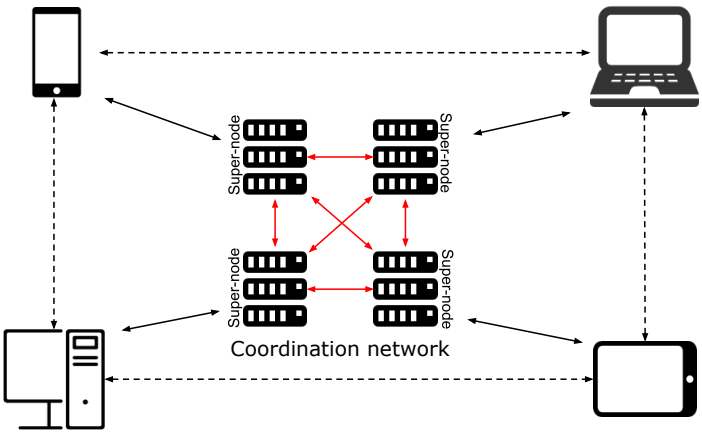


Figure 9.8: Super node based P2P framework approach

One way to improve this situation is to decentralise the coordination network by using a P2P network made of super-nodes, as illustrated in Figure 9.8. Super-nodes are nodes with a good connectivity, low latency and high availability. It is a well known concept and, for example, is also employed by older versions of the Skype messenger [Baset and Schulzrinne, 2006], before it was switched to a cloud-based system. Each mobile device can have connections to one or multiple super-nodes. In contrast to the previous approach, the mobile devices are not part of the super-node network, therefore mobile devices are not subject to deviating latencies in the overlay network. The establishment of direct connections between the mobile devices is comparable to the previous approach, with the advantage that multiple access nodes to the coordination network can be set-up, for example close to crowded places, still, the access nodes are interconnected in terms of computing unit discovery.

The coordination network is built for easy deployment using the Java programming language and instantiating the TomP2P [TomP2P, 2014] framework. TomP2P is a robust Java-based Distributed Hash Table (DHT) implementation with advanced features like NAT hole punching. Mobile devices use WebSockets [Fette and Melnikov, 2011] to build a connection to the coordination network, and register their availability and connection information. They can choose to accept incoming WebSocket connections, or setup a WebRTC endpoint. For WebSocket endpoints, URL and certificate information is saved in the coordination network. For WebRTC endpoints, the coordination network is used as a signalling channel to initiate and establish the WebRTC connection.

From a high abstraction level it seems to be a step back regarding flexibility and fail-safety. In fact, the duties on the mobile devices are relieved and dependencies are removed:

- Mobile devices only need a single connection to the coordination network. In the previous approach at least a centralised signalling channel and other mobile devices acting as access nodes were required for the initial connection to the P2P overlay network.
- Mobile devices are not responsible for routeing requests, therefore latency of the overall overlay network does not depend on the connection conditions of single mobile nodes in the network.

Best practice for the deployment of this approach, as illustrated in Figure 9.9, is to move single access nodes of the coordination network close to the mobile devices, but also provide global access nodes, to enable connectivity for other, not latency sensitive computing units. This deployment approach also tackles the issue of finding geographically proximate nodes. Each access node is connected to proximate nodes, finding other computing units is then reduced to selecting available nodes of the same access node. Still, the capabilities to use computing units from other access nodes is maintained, due to the interconnection of the coordination network in a P2P manner.

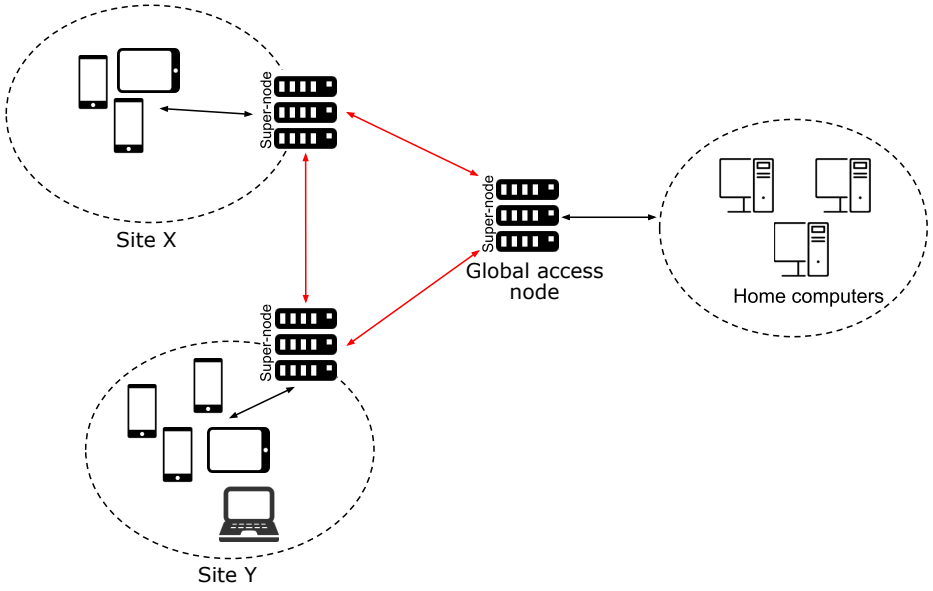


Figure 9.9: Decentralised coordination network deployment

9.2.6 Isolation

One aspect not tackled so far is the isolation of multiple executions from different users on the computing units. In this case isolation basically aims at two goals:

1. Protect assets of the computing unit owner, such as data or components of the computing unit, from unauthorised access by the offloaded applications.
2. Protect the offloaded computation and data from access by the owner of the computing unit.

The server-side of the offloading framework is realised using two distinct approaches, which require different isolation solutions highlighted in the following chapters.

9.2.6.1 Isolation on Mobile Devices

On mobile devices operated as offloading servers, offloaded applications are executed in web views. Web views are web browsers which can be embedded in native applications on the mobile device and still provide all features and security mechanisms of web environments. Offloading servers on mobile devices are running as top level applications in web views. A single offloading server can accept multiple offloading requests simultaneously, still they are not allowed to access each other's data or the device owner's data.

The device owner's assets are already well protected by employing the web view. No unauthorised access to the device owner's data is possible. To also protect the offloading operations, each operation is launched in a separate *iframe*[15]. Basically, spawning a new *iframe* is straight forward, as illustrated in Listing 9.3, by adding an *iframe* element to the DOM-tree of the root frame.

```
<iframe id="offloading id" src="URL to the source code">
</iframe>
```

Listing 9.3: HTML iframe syntax

[15] https://developer.mozilla.org/en/docs/Web/HTML/Element/iframe

The issue with this approach is that the source code, executed in the *iframe*, can only be specified as an URL, but the source code is downloaded from one of the repositories and finally assembled to JavaScript on the server device. Therefore, a dynamic URL needs to be created, which resolves to the modified JavaScript source code. This is achieved using the HTML *File API*[16] as illustrated in Listing 9.4. The result of this operation is a URL of the form *blob:BLOB_URL/UUID*, which is a unique URL for each call to this method. This URL can be used as the source URL for the created *iframe*.

```
var srcUrl = URL.createObjectURL(new Blob([src]));
```

Listing 9.4: HTML File API usage

Another characteristic of *iframes* is that they act like a separate web environment instance, and can only communicate with the main application (the parent frame) through a defined and controlled channel using *postMessage*[17]. This communication channel is used to transfer the initial data and arguments to the offloaded application part, and to transfer the results to the parent frame. This way each executed offloading operation is executed in an isolated environment.

9.2.6.2 Isolation on Dedicated Computing Units

On dedicated computing units used for offloading, separate processes are started for each offloading operation. On unprotected systems, these processes basically can access all files and resources. Depending on the used operating system different measures need to be applied.

Recent versions of the Windows operating system support *AppContainers*[18] which can be used to activate isolation capabilities. Commercial products also exist, which hook into the operating system and introduce an intermediate layer to capture relevant system calls. On Linux operating systems, sandboxing techniques are more advanced and integrated into the system. By executing processes as a restricted user, the application already has small chances of harming the system. More sophisticated solutions like *AppArmor*[19] or *firejail*[20] enable to specify on an application basis, allowed actions, and what applications are allowed to see on the system. For even higher isolation levels, virtualisation techniques like KVM can be used to start a virtualised environment and launch the application in the virtualised environment.

9.2.7 Realisation

The implementation is not only in a concept phase, but utilises the full potentials of the environment. Screenshots of selected scenarios are illustrated in this chapter. First, to illustrate the broad applicability for different devices and vendors, the supported and realised technologies are listed.

Focussing on mobile devices, there is no need to differentiate between different architectures. For POWER, support for web technologies like WebSockets and WebRTC is essential. Therefore, a distinction between web technology-enabled platforms and not-yet-enabled platforms is required. Currently only Android provides the necessary feature-set, iOS is still missing support for WebRTC. The realisation can take different shapes: It can execute in the browser, using web-based cross-platform frameworks like Apache Cordova or use Dart-based cross-platform frameworks like Flutter. Another possibility, which is not considered by the implementation, is providing a native application with an embedded Dart compiler or virtual machine. On desktop systems, similar possibilities exist regardless of the utilised operating system or architecture: It can execute in the browser, using cross-platform frameworks like

[16] https://www.w3.org/TR/FileAPI/
[17] https://developer.mozilla.org/en-US/docs/Web/API/Window/postMessage
[18] https://msdn.microsoft.com/en-us/library/windows/desktop/mt595898(v=vs.85).aspx
[19] http://wiki.apparmor.net
[20] https://firejail.wordpress.com/

Electron[21] or Chrome applications. On desktop systems, as well as on server systems without a graphical user interface, POWER can directly be executed using the Dart VM command line. A summary of these realised capabilities is provided in Table 9.1.

	Mobile web-technology enabled platforms				Desktop/Server Platform			
	Browser	Web technology-based cross-platform frameworks	Dart-based cross-platform frameworks	Native	Browser	Cross-platform frameworks	Chrome application	Dart VM
Support provided by current implementation								
Considered but not featured by current implementation								
Not considered/out-of-scope								

Table 9.1: Realised capabilities of provided implementation

The following Figure 9.10 and Figure 9.11 illustrate the different shapes of POWER in server mode. The figures only show a fraction of the possible realisation. A complete picture is shown in Table 9.1. On Android, POWER is operated as a standalone application and is connected to one of the coordination nodes, waiting for incoming offloading requests. On the desktop environment, POWER is directly launched from the browser, fulfilling the same purpose. The different user interface representations illustrate the adaptability. Depending on the user's requirements different levels of details can be shown to the user. Security aspects and enabling technologies are discussed in Part IV of this thesis, still as a

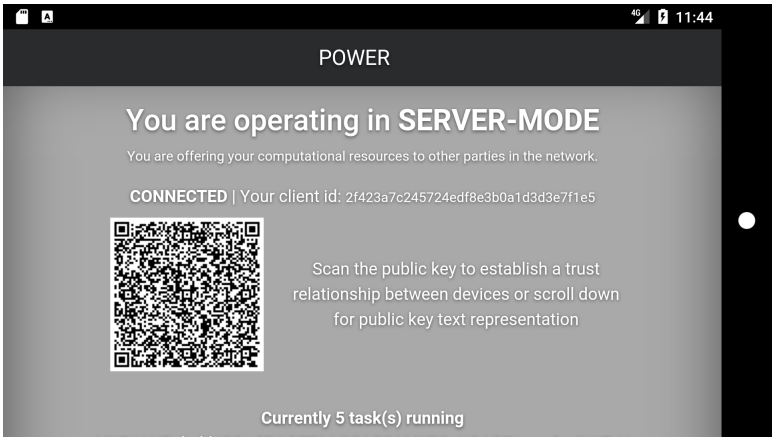


Figure 9.10: POWER-server mode on Android in application mode

baseline the application automatically generates a public-private RSA key pair, used to authenticate at other devices. The client ID is derived from a user's public key (in the default case). This way, a user can verify if he really is connected to the device with a specific client ID, if the public key was exchanged beforehand. This is enabled by displaying a QR code containing the public key to be scanned by e.g. client devices. The command-line version targeted at server environments without graphical user interfaces are not illustrated here. They basically offer the information as a text-based version.

On the other side of the spectrum are the client applications, requesting computational power from the network. Figure 9.12 and Figure 9.13 illustrate two representations of the PBKDF2 application, also used as part of the performance evaluation in Chapter 9.3. The application performs the PBKDF2 algorithm with user provided attributes. Optionally, a node for the offloaded execution can be specified. In Part IV measures are specified to outsource this decision to trusted authorities. The utilisation of the framework is completely transparent to the user. It tries to establish a connection to one of the proximate computing units, acquired via the coordination network, and outsources the computation. This again does not represent all realised possibilities. They are listed in Table 9.1. Also command-line applications,

[21] https://electron.atom.io/

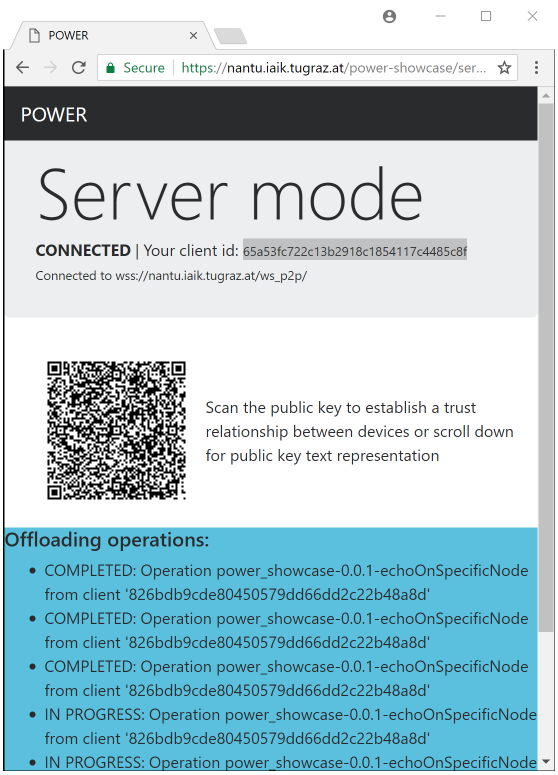


Figure 9.11: POWER-server mode on Windows in browser mode

operating on the Dart VM, can offload their computations, maintaining the same properties for the user, like complete transparency.

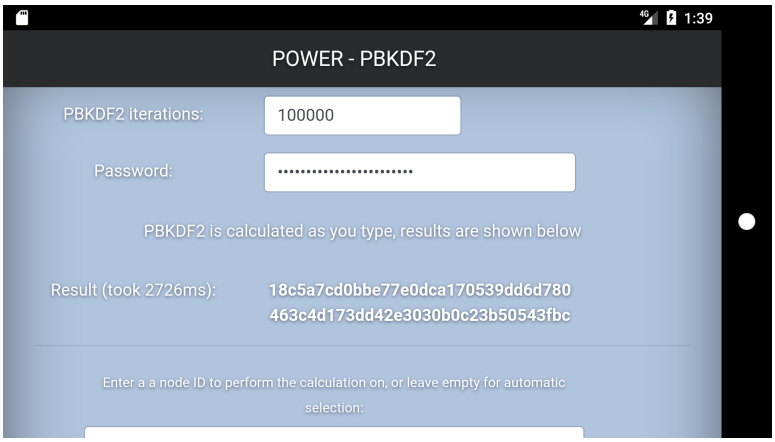


Figure 9.12: POWER-enabled application on Android in application mode

9.3 Framework Evaluations

In this section the POWER framework is evaluated in terms of energy consumption and performance improvements. The evaluations are published in [Reiter and Zefferer, 2016]. For the evaluations, the key aspects of two applications are executed in an offloaded way:

- A raytrace application used to render images.

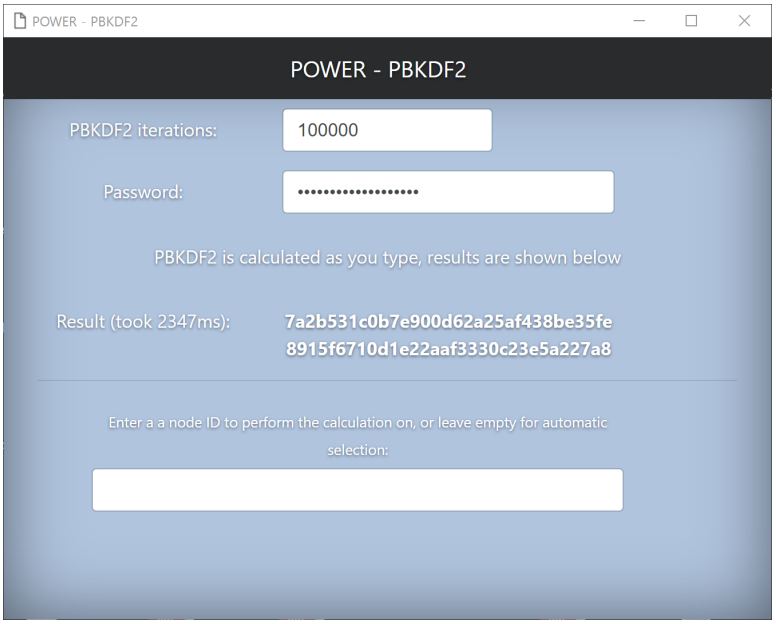


Figure 9.13: POWER-enabled application on Windows in application mode

- An application making use of PBKDF2, a password-based key derivation function.

For the energy consumption evaluation, the raytracer application is used to clearly show the impact on the different mobile device components. The performance evaluation is performed using both applications. The evaluations solely focus on energy and performance impacts of the offloading process, without going into detail on the used communication technologies as introduced in Chapter 9.2.5. Regardless of the used technique, the final result is always the same: Mobile devices and computing units are directly interconnected, either using WebRTC or using WebSockets. Tests showed that the impact of the concrete used approach can be neglected.

These evaluations show the impact, compared to local executions, of applying the offloading framework to applications in terms of energy consumption and performance. Providing a comparison to previous frameworks following a very different execution model (e.g. MCC) is problematic in multiple ways. To get meaningful results an equal technological basis is required. Existing frameworks are tailored to specific environments, operating systems and programming languages. The proposed realisation provides flexibility in terms of the execution environment and operating system. For the implementation, a programming language is chosen that is available on a majority of the target platforms. The tenor of this thesis is to focus on MCC/HMCC/HMEC improvements in the field of security, flexibility and interoperability enabling a re-usability of the technological foundation contributed by other frameworks, especially in terms of client side offloading techniques. Therefore, due to the limited expressiveness of comparing performance and energy consumption improvement results of existing frameworks with the proposed approach, the comparison in intentionally omitted.

9.3.1 Test Setup

The test setup, used for the evaluations, is illustrated in Figure 9.14. The test setup comprises three computing units, whereas two are directly accessible via public IP addresses, and one is behind a NAT where no incoming connections are allowed. A STUN/TURN server is deployed to allow a reliable discovery of a server's public IP address and to enable relayed connections. The evaluations are performed using multiple resource constrained devices, which require assistance from computing units: Different Android 5 devices were used as mobile device platforms, furthermore Windows and Linux PCs running the applications in Firefox and Chrome were evaluated. Other mobile platforms like iOS and Windows

Phone are still constrained in their support for the used new web technologies, therefore an evaluation was not possible.

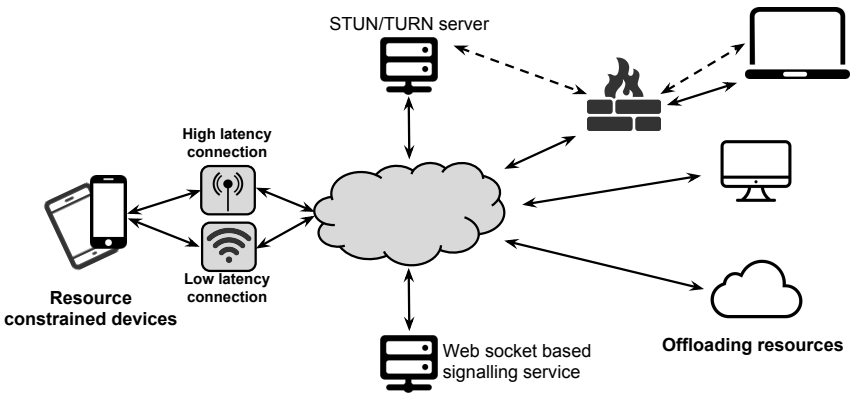


Figure 9.14: Energy consumption and performance impact test setup [Reiter and Zefferer, 2016]

9.3.2 Energy Consumption Evaluations

High energy consumption is one of the main drawbacks of today's mobile devices. Running applications with high performance requirements on the mobile device draws a lot of energy and inherently reduces battery lifetime with the result that users need to charge their phones multiple times a day on heavy usage. This chapter highlights the achieved energy consumption improvements by applying the proposed HMEC framework.

This evaluation is only relevant for battery powered devices and therefore is only performed on the mobile device. In contrast to the performance evaluation, the connectivity aspect is added to the evaluation. As the evaluation will show, it makes a significant difference which technology is used to interconnect devices (3G or Wifi). To measure the energy consumption for mobile devices multiple approaches can be used:

- Hardware-based approaches directly attach analysers between the mobile device and its battery, but have significant drawbacks in mobility.
- Model-based approaches, as proposed by Zhang et al. [2010], can be used to generate software models to estimate the power consumption of mobile devices. Unfortunately, models greatly differ between different mobile device models.
- Recent Android versions record battery-related events, which can be analysed with the battery-historian tool[22]. This allows to reconstruct a battery consumption estimation on application and component level.

The energy consumption evaluations rely on the battery-historian approach, as it seems to be the most flexible. To get accurate results, the mobile device was set to a known state before the evaluation: battery was fully charged, screen left on at the lowest brightness level, all synchronisation services, GPS and other background tasks were deactivated.

The evaluations are performed using the raytracer application with the performance profile and energy saving profile. Both profiles were evaluated with computing units remotely connected via Wifi connections with low latencies and via 3G connections with latencies around 220ms. The expectations on the results can be summarised as follows:

[22] https://github.com/google/battery-historian

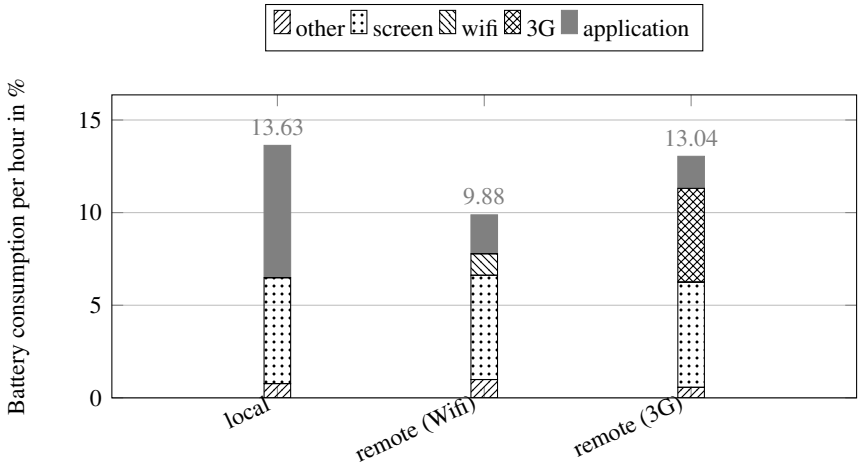


Figure 9.15: Energy consumption evaluation using the performance profile stacked by component [Reiter and Zefferer, 2016]

- Each device has a baseline energy consumption of applications and services running in the background. This energy consumption is not influenced by the offloading framework.
- Due to ever increasing screen sizes, it is expected that the mobile device's screen is the largest energy consumer.
- It is expected, as research already shows [Cuervo et al., 2010] that offloading via 3G connections requires considerable more energy than offloading via Wifi.
- It is anticipated that the performance profile will not save energy, but the energy consumption shifts from the application to the communication technology.

The results for the performance profile are shown in Figure 9.15. Each vertical bar shows the major energy consumers for this test run. In the case of local execution, it is shown that the screen is one of the main energy consumers. The primary goal of the performance profile is to make vast improvements in performance without considering energy consumption. Still, beside performance improvements, the results show tremendous improvements in energy consumption too. Using the Wifi connection and this specific application, the energy consumption per hour was more than cut in half, including energy consumption of the Wifi component. The 3G connection requires considerable more energy, still energy consumption was reduced. These energy consumption values already include energy consumed by the framework and all background tasks introduced by the framework. The expectations apply in terms of baseline energy consumption, screen energy consumption, and the energy consumption gap between 3G-based and Wifi-based offloading. Regarding the overall energy consumption, the results even outperform the expectations. The offloading operations have a lower energy consumption than anticipated (even in the 3G case) and achieve remarkable energy saving effects.

The evaluation results obtained using the energy saving profile are shown in Figure 9.16. Using the energy saving profile, the overall energy consumption of the application was cut from 7.1% per hour for local executions to 0.3% per hour for offloaded executions using the Wifi connection and 1.16% per hour for offloaded executions using the 3G connection. The expectations also apply for the energy-saving use case, the application's energy consumption can be minimised tremendously.

9.3.3 Performance Impact Evaluations

The impact on the performance of employing the POWER framework is analysed based on two applications:

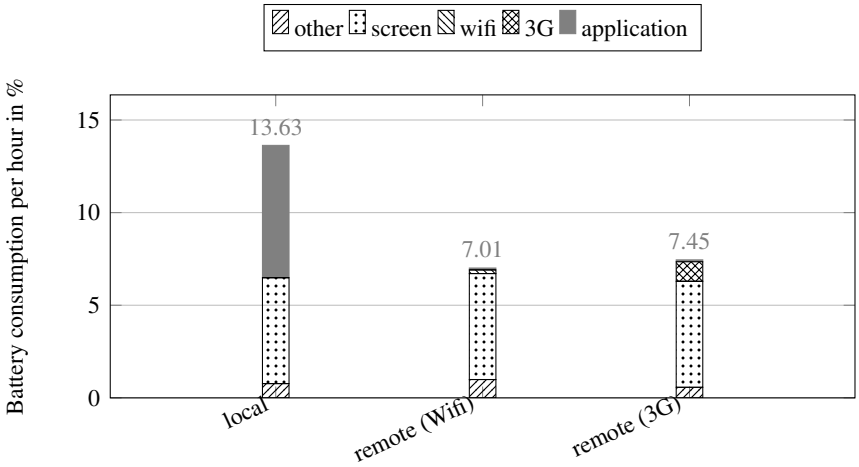


Figure 9.16: Energy consumption evaluation using the energy saving profile stacked by component [Reiter and Zefferer, 2016]

- A raytracer application is modified to offload the computation of individual scenes to remote computing nodes.
- A PBKDF2 application is modified to offload the password derivation to trusted (refer to Part IV of this thesis) computing nodes. The strength of keys, derived from passwords is heavily impacted by the number of performed iterations which is limited on mobile devices.

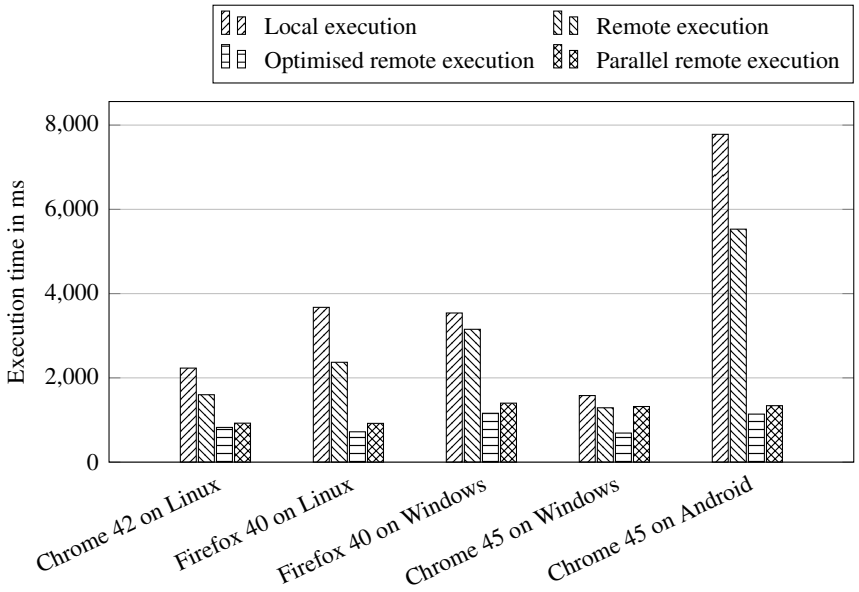


Figure 9.17: Raytracer performance for rendering a 256 pixels by 256 pixels image [Reiter and Zefferer, 2016]

The raytracer application does not utilise any optimisations from the field of 3D graphics. It depends on a scene description as input parameters, and returns the rendered bitmap. The following information needs to be transferred to the remote side on offloading: The scene and application state information is encoded in about 1165 bytes (depending on the actual provided information), compressing this information can shrink it to about 631 bytes. The scenes were rendered to bitmaps comprising 65535 pixels and 262144 pixels. The results were obtained by rendering the same image 200 times and calculating the average required time. To minimise the impact of latencies, the devices were connected with low latency

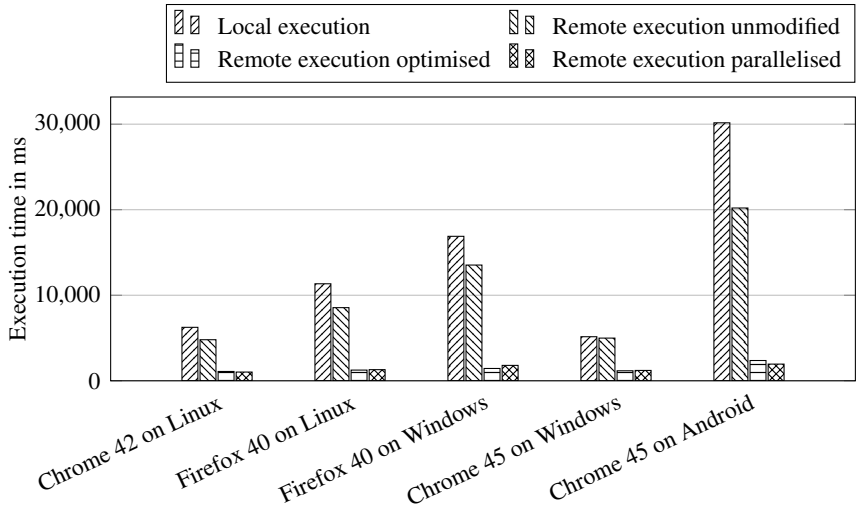


Figure 9.18: Raytracer performance for rendering a 512 pixels by 512 pixels image [Reiter and Zefferer, 2016]

(single digit) Wifi connections. The expectations on the framework in this regard can only be formulated on an abstract level: enable to increase the performance of mobile applications. Actually, in this dynamic environment, the execution performance depends on various factors like connection latency, connection bandwidth, available processing power of the computing unit, and suitability of a particular application to apply the offloading framework. Actually, the framework is built in a way to plug in different offloading profilers and decision engines, hence, the goal is not to compete on a profiler or decision engine level with e.g. MAUI [Cuervo et al., 2010] or ThinkAir [Kosta et al., 2012], but enable to extend these frameworks with enhanced security features. The plug-able structure enables to reuse already existing concepts.

The results for the 65535 pixels rendering process are shown in Figure 9.17. As illustrated, local only executions can take up to 8000ms on Chrome for Android, which makes this kind of application unusable. Without applying any optimisations to the application, the execution time was already reduced by about 30% to 5500ms. The application is developed in the Dart programming language, and the computing units are dedicated units which directly execute Dart code. This explains the performance gains for desktop environments, because the utilised browsers are only executing the transformed JavaScript code.

The next step was to slightly optimise the application. Currently, the application returns a blunt list of pixels. Just by encoding the returned picture as PNG further reduced the execution time to 1140ms. This is a performance gain of more than 700% compared to the original local execution. To further improve the results, the execution was parallelised, by rendering multiple parts of the image at different locations and consolidating the image locally. As it turned out, the introduced overhead for the local consolidation overwhelmed the performance gains of the parallel execution.

Similar performance gains can be observed for the 262144 pixels image in Figure 9.18. On mobile devices the execution time was reduced from more than 30 seconds to 1950ms.

The evaluation of the PBKDF2 applications requires a trusted offloading computing unit. To get a realistic scenario the computing unit behind the NAT was considered as home computer, trusted by the user. Generally, there are no common guidelines on the number of iterations, to achieve a sufficient level of security. A rule of thumb is to select a number of iterations which takes about one second to compute on the target device. It is clear that this number will be lower on mobile devices than on dedicated computing units. Therefore, the goal of employing HMEC in this use case is to increase the overall security. Performance values were collected for 1000, 10000, and 100000 iterations. The results are shown in Figure 9.19. The defined rule of thumb is only satisfied by desktop browser for up to 10000 iterations with local executions. Mobile browsers already need 3253ms in the local execution scenario

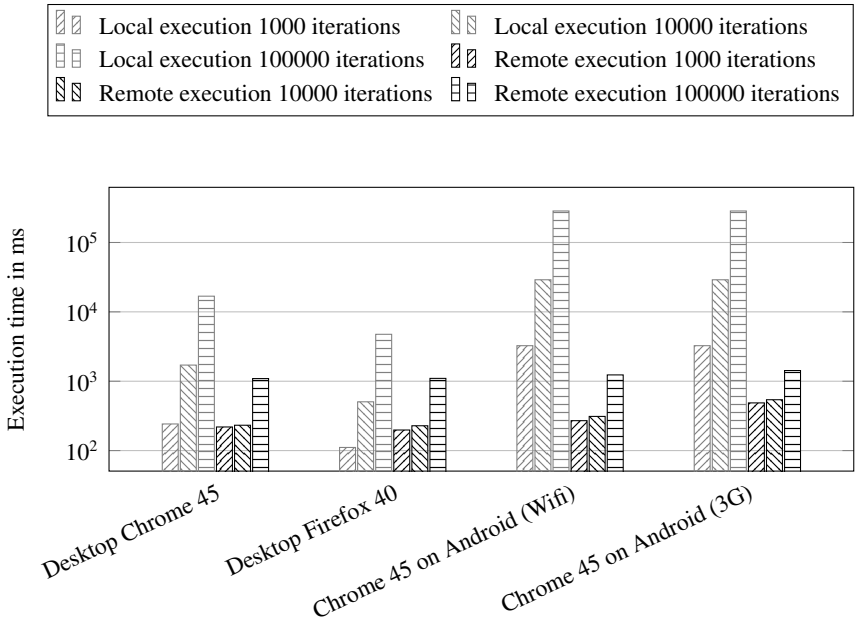


Figure 9.19: PBKDF2 performance graph on logarithmic scale [Reiter and Zefferer, 2016]

for 1000 iterations. By employing the offloading approach, 100000 PBKDF2 iterations can be performed in mobile device browsers in just 1426ms.

9.4 Chapter Conclusions

In this chapter an implementation was developed based on the refined architecture from Chapter 8 to prove its feasibility and capabilities. The requirements and goals as defined in the architectural phase are translated to clear targets on the implementation level, still following the central theme of achieving a *lightweight*, *flexible*, and *interoperable* solution. To closely follow these targets, the boundaries of the environment have been scrutinised. It is not feasible to create a solution which covers the whole landscape of available applications, devices and architectures, and still stick to the central theme. Therefore, the operation environment was defined and aims at the sweet spot between reaching the defined targets and being compatible with existing applications: At its current stage the framework does not focus on native applications of any platform, but concentrates on applications built on web technologies like web applications or native-like cross platform applications. The implementation focuses on filling the identified gaps, instead of reinventing already established and well-researched topics.

The implementation is evaluated with a focus on the energy consumption and performance impact of the solution. Although the results are depending on the concrete application and its structure, the achievable improvements are clearly shown. For the applications under test, the overall energy consumption was more than cut in half. Using the performance profile, still notable energy savings were achieved, and also performance improvements of up to 700% for selected applications.

This implementation serves as the baseline for a gradual extension and improvement regarding users' privacy and security in the following Part IV of this thesis.

Chapter 10

Part III Conclusions

Due to the increasing migration of daily tasks from desktop computers to mobile devices like smartphones and tablets, users' demands are increasing in terms of processing power and battery lifetime. the current predominant situation either requires users to recharge their mobile devices multiple times a day on heavy load, or rely on cloud services. For interactive, latency sensitive applications a high bandwidth and low latency Internet connection is required. Furthermore, cloud services often only serve predefined tasks like save data or perform pre-agreed calculations. A more flexible approach is required to react to changing environments like the current battery level, connection status or current location.

Current approaches to improve the situation enable applications to determine parts at runtime or at compile time, which could be executed remotely, also referred to as offloaded execution. Evaluations of these existing approaches show that, under ideal environmental conditions, high energy savings and vast performance improvements can be achieved. This is just a first step towards a complete solutions. Existing solutions are often statically linked to single cloud solutions and therefore the profit of applying offloading technologies is limited.

This part of the thesis starts by identifying the still existing gap from existing approaches to solutions. Several gaps and final goals are identified: *lightweight*, *flexible*, and *interoperable*. An architecture which fills the identified gaps is developed, also taking the already existing approaches into account. The architecture specifically takes into account a high fluctuation of computing units used for the offloading operation and their organisation. To prove the feasibility of the architecture, a solution is developed based on the enhanced architecture. Unlike existing approaches, it does not only allow dedicated computing units, but enables all participating nodes to consume and provide computational power in the system. The computing units are organised in a P2P like structure, where multiple approaches were evaluated to find the most adequate in the defined operation environment. The evaluation results show that both, energy savings and vast performance improvements are possible, also without relying on statically linked and prepared cloud services.

The developed solution realises the envisioned fulfilment of the defined security requirements. This implementation serves as the first approach realising a dynamic offloading approach, which considers all the defined high-level functional requirements. The missing links to finally consider the security requirements, not only on architectural level, are discussed in the following Part IV.

Part IV

Trust, Security and Privacy

Chapter 11

Processing Sensitive Data

The previous parts of this thesis established a sound model defining an improved architecture based on analysing existing offloading frameworks, providing an implementation based on this architecture, and evaluating the solution. The evaluations clearly focussed on two key aspects of offloading frameworks: energy saving effects and performance impacts. Those are indisputable important aspects of HMEC frameworks.

Up till now, users with the requirement of protecting their privacy of processed data and workflows were unable to make use of HMCC/HMEC concepts. Already existing frameworks focus on technical procedures and algorithms concerning the offloading operation: How to identify parts of an application which should be considered for offloading, how to perform the offloading process on a technical level, and how to decide if offloading is beneficial under given device and environmental conditions. Security aspects are not considered by any of the analysed frameworks in Part II of this thesis.

This chapter settles the scene to establish a trust model for HMEC frameworks by (a) identifying the users requiring protection of their privacy and data and (b) introducing privacy protection and trust establishing means from other, related domains. Following this chapter, the overall goal of this part is to transfer the introduced and already well-established means to the domain of HMEC.

Two different user groups of HMEC frameworks can be identified:

- **Private users:** In the recent years, even large amounts of private users gained data security and data protection awareness. Many private users understand that revealing their private data to untrusted entities is a risk, and data may be sold to third parties for e.g. targeted advertising campaigns. Even large and well-established companies analyse private data to generate profit (e.g. Google Gmail[1] just to name one example). However, applications targeting private users often do not involve personal data, like gaming applications. Still, computing units should behave as expected, but no data can get lost in this case.

- **Corporate users:** A different picture is drawn for corporate use cases and users. Corporation-managed applications have always access to corporation data, corporation workflows, or the corporation network. Corporations need full control of the processing entities in terms of particularly selecting single allowed nodes or whole providers. Furthermore, different tasks might process data with different levels of confidentiality which should also influence the selection of the computing unit.

Many corporations already have systems in place to control accesses to corporation controlled endpoints or from corporation controlled applications to outside endpoints. Extensible Access Control Markup Language (XACML) [OASIS, 2013] is the de-facto standard language in the industry to define access control policies following an Attribute-based Access Control (ABAC) approach. XACML is

[1] https://www.theguardian.com/technology/2014/apr/15/gmail-scans-all-emails-new-google-terms-clarify

defined in an open way and actually can be used in various different scenarios. The remainder of this chapter is dedicated to provide the required details on the XACML technology as well as state-of-the-art approaches of enforcing the defined policies in Chapter 11.1. These approaches foresee an explicit integration of the enforcement infrastructures into service providers and service consumers. This was picked up by the H2020 SUNFISH[2] project and was extended in the direction of data usage control, fostering a transparent integration of the enforcement infrastructure in existing and new applications. An overview of this extended enforcement infrastructure and the contributions are discussed in Chapter 11.2.

11.1 Extensible Access Control Markup Language and Enforcement (XACML)

XACML is a security policy language developed and maintained by the OASIS[3] consortium. On the one hand it defines a policy language model to express a multitude of different security requirements, on the other hand, it defines conceptual entities to guide developers on how to actually integrate the defined concepts. In the following sections, a concise overview of these concepts is provided.

11.1.1 Policy Language

XACML specifies a format to provide policies in Extensible Markup Language (XML) format. The goal of specifying and enforcing policies is to get a decision if a certain action is allowed. With policy languages, this authorisation is decoupled from the actual program logic, enabling a centralised management of authorisations and adaptations without modifying application logic.

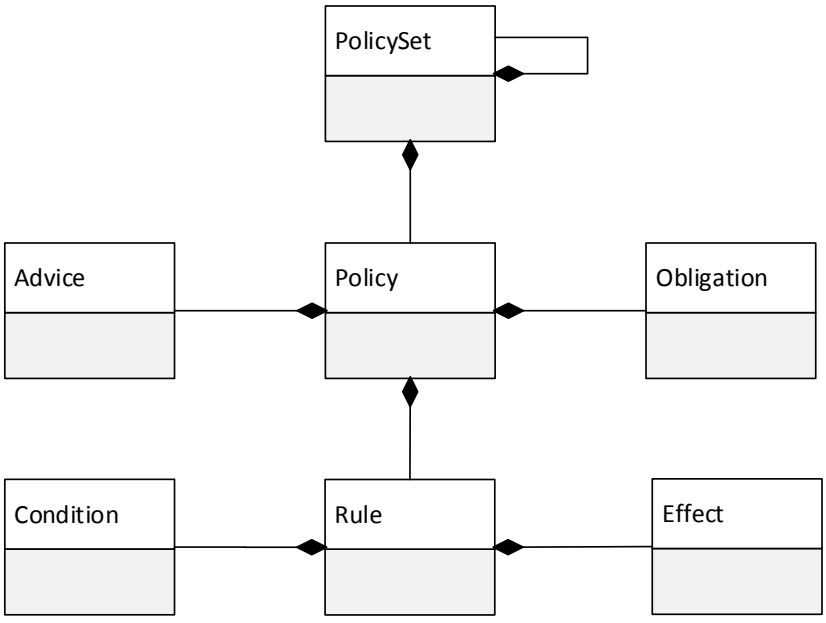


Figure 11.1: XACML entities

To achieve this, XACML is based on an ABAC approach. Policies are based on the comparison of provided attribute values, to the values (or variables) embedded in the policies. The comparison functions can be defined flexibly, referring to values of the provided context, the environment, or can even be completely customised. Due to the flexibility of the ABAC approach, a profile was defined to

[2] http://www.sunfishproject.eu

[3] https://www.oasis-open.org

also map Role-based Access Control (RBAC)[4] approaches to XACML. XACML fosters reusability of the defined policies by enabling to link between them. In the specification the basic entities *Rule*, *Policy* and *PolicySet* are defined. Each entity has a unique identifier for reference. The relations between these entities are illustrated in Figure 11.1. Rule entities are the most atomic units and provide the actual authorisation logic. The policy entity, in contrast, combines multiple rules for evaluation by the XACML engine. The policy set entity again combines multiple policies (and other policy sets). This way, policies and policy sets can be cross-referenced, avoiding multiple definitions of the same elementary policies.

At the lowest level, the rule-level, each entity additionally defines the effect of a matching policy by either explicitly permitting or denying access. Moreover, conditions can be added to exactly define matching policies. At the policy level, two more entities, obligations and advices, can be defined. They can be used to define obligatory or optional actions to be performed at policy enforcement.

This multi-levelled approach enables a fine-grained specification of policies and a clear hierarchical structure, but also leads to inconsistencies and corner cases like no matching policies and multiple policies yielding different results. To combine the evaluation results of multiple rules, policies, or policy sets, XACML defines combination algorithms to specify how to deal with different results. In case no matching policies are defined a dedicated result value *NotApplicable* is defined. Particular combining algorithms can define further constraints on the policies. In case they are not satisfied the evaluation can result in the value *Indeterminate*.

The following Listing 11.1 provides a simple policy example, showing the overall structure of XACML policies. To improve the readability, XML namespaces and prefixes are removed and all rules are embedded into the root entity without using the reference capabilities. The policy matches on requests where the service attribute has the value *ExampleService* with the goal to only allow access from 9 am to 6 pm. The policy contains two rules:

- The first rule *DefaultDeny* denies all accesses.
- The second rule evaluates an environment variable containing the current time of the day and only permits access from 9am to 6pm.

During evaluation, the first rule always evaluates to *Deny* and the second rule evaluates to *Permit* or *Deny* depending on the current time of the day. Finally, it is up to the combining algorithm to decide how to combine the rule evaluation results. In this particular example, access is denied per se and only allowed in certain scenarios. Therefore, a rule allowing access needs to overwrite the denied access.

```
<Policy PolicyId="SamplePolicy1"
 RuleCombiningAlgId="...:rule-combining-algorithm:permit-overrides">
 <Target>
  <AnyOf>
   <AllOf>
    <Match MatchId="...:function:string-equal">
     <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string">
         ExampleService</AttributeValue>
     <AttributeDesignator
      AttributeId="...:attribute:id"
      Category="...:attribute-category:service"
      DataType="http://www.w3.org/2001/XMLSchema#string"
      MustBePresent="true"/>
    </Match>
   </AllOf>
  </AnyOf>
 </Target>
 <Rule Effect="Deny" RuleId="DefaultDeny">
  <Target/>
```

[4] https://docs.oasis-open.org/xacml/3.0/xacml-3.0-rbac-v1-spec-cd-03-en.html

```
  </Rule>
  <Rule Effect="Permit" RuleId="9to6">
   <Condition>
    <Apply FunctionId="...:function:and">
     <Apply FunctionId="...:function:time-greater-than">
      <Apply FunctionId="...:function:time-one-and-only">
       <AttributeDesignator
        AttributeId="...:environment:current-time"
        Category="...:attribute-category:environment"
        DataType="http://www.w3.org/2001/XMLSchema#time"
        MustBePresent="true"/>
      </Apply>
      <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#time">09:00:00Z</
          AttributeValue>
     </Apply>
     <Apply FunctionId="...:function:time-less-than">
      <Apply FunctionId="...:function:time-one-and-only">
       <AttributeDesignator
        AttributeId="...:environment:current-time"
        Category="...:attribute-category:environment"
        DataType="http://www.w3.org/2001/XMLSchema#time"
        MustBePresent="true"/>
      </Apply>
      <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#time">18:00:00Z</
          AttributeValue>
     </Apply>
    </Apply>
   </Condition>
  </Rule>
</Policy>
```

Listing 11.1: XACML sample policy

By further hierarchically structuring the rules, policies, and policy sets, also the combination algorithms are hierarchical structured and enable a high re-usability of the defined policies. As an example: the *9to6* rule could be modelled as a separate policy, and could be used by different policy sets to enable access from 9 am to 6 pm, by only providing the reference to the defined policy.

11.1.2 Enforcement Model

The adherence to security needs requires two fundamental obstacles to overcome in the first place: A definition language to express security requirements and means for their enforcement. A big picture and the basic workings of the policy definition language is provided in Chapter 11.1. Besides, the XACML specification also defines entities and models on how to enforce the defined policies. The following abstract entities are defined, with their interactions as illustrated in Figure 11.2:

- The *Policy Administration Point (PAP)* manages the available policies and offers interfaces to retrieve them.
- The *Policy Information Point (PIP)* acts as a source for additional attributes required in the policy evaluation process, e.g. additional environmental parameters, additional information of the accessed resource, or the accessors.
- The *Policy Decision Point (PDP)* loads the available policies and evaluates policy decision requests against the current set of policies. In case policies are referring to not provided attributes, they are requested from the PIP.

- The *Policy Enforcement Point (PEP)* acts as the application entry point and issues policy decision requests to the PDP and later on enforces the received decision responses.

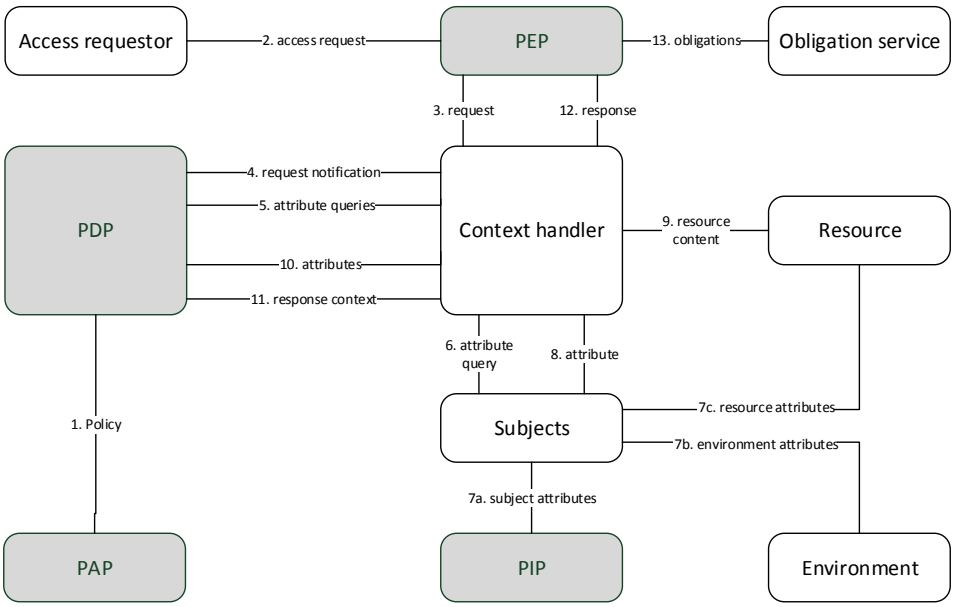


Figure 11.2: XACML enforcement data flows[OASIS, 2013]

The XACML specification only provides interfaces for the PDP. The other components are loosely defined on an interaction level, without going into detail on the actual used formats, protocols, or technologies. In Figure 11.2 it is assumed that the PAP already contains all relevant policies. The numbers in the figure illustrate the sequence in the startup, evaluation, and enforcement phase. They correspond to the numbers in the following listing:

(1) At initialisation of the XACML components the PDP has access to all the defined policies at the PAP. Either the PDP loads all available policies at startup, or the PAP provides an interface where policies can be retrieved selectively.

(2) An external entity is requesting access to a resource at the PEP. The PEP can be realised as a separate component, or as in integral part of applications. The XACML specification leaves the concrete realisation open to implementers.

(3) The PEP sends an access request to the context handler and may already include attributes describing the subject, resource, action, environment, or custom attributes.

(4) The context handler is an intermediate entity translating the decision requests format to the XACML conform format and interfaces with the PIP to collect missing attributes. It then sends the transformed XACML conform request to the PDP.

(5 - 10) The PDP identifies and requests attributes required for policy identification and evaluation from the context handler. The context handler identifies the source of requested attributes and returns the attributes.

(11 - 12) The PDP evaluates the policy decision request against the retrieved policies and attributes and returns the result to the context handler, which translates the decision back to the PEP supported format.

(13) The PEP applies the contained obligations and advices and returns the decision to the requester.

Using these baseline approaches, a data security enforcement framework can be specified fulfilling the requirements of various applications. Overall, XACML is very loosely specified and requires redefinition and detailing for particular use cases. The enforcement approaches and techniques are not specified at all. The specification only mentions the existence of enforcement components, but without providing any details on best practices.

11.2 SUNFISH Data Security Approaches

The H2020 SUNFISH project[5] aims at developing and integrating software that enables secure cloud federation as required by the Public Sector. This requires the definition and realisation of a sound enforcement model and the extension of XACML to also enable a data usage control, beyond authorisation. A requirement from use case owners concerning the data security enforcement models was to enable the re-usability of already existing applications, without explicitly integrating XACML entities but still benefiting from the defined approaches.

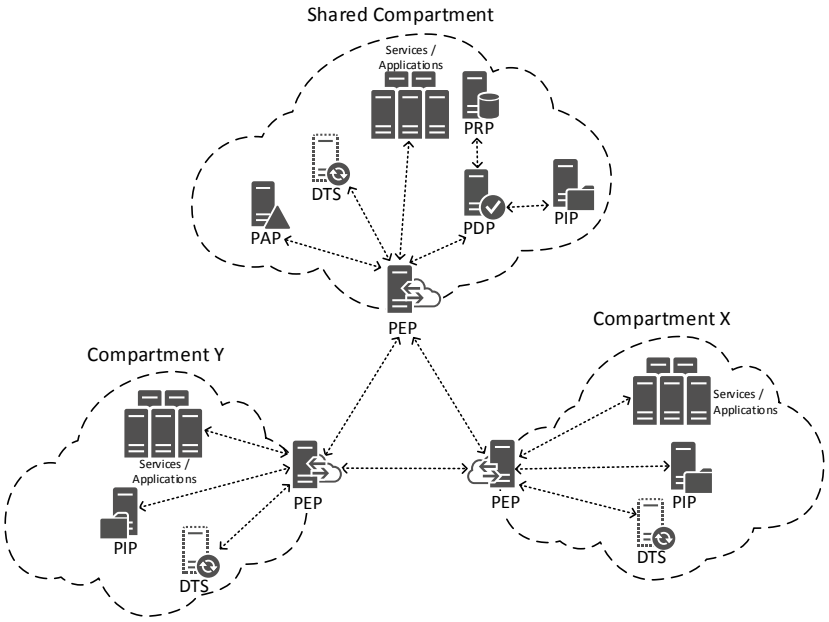


Figure 11.3: SUNFISH enforcement model

Existing XACML implementations realise the authorisation logic (PDP), a corresponding policy store (PAP), and define ways of gathering additional attributes. The PEP implementation is left to application integrators, by intention. This requires XACML awareness of services and application and explicit interactions to the XACML infrastructure as foreseen by the specification. To fit into the SUNFISH approach a redefinition and detailing of the enforcement model is required as illustrated in Figure 11.3 and published by Suzic and Reiter [2016] with the goal of also giving legacy applications the benefits of XACML.

The SUNFISH data security enforcement model is based on different security compartments. Each compartment contains relevant parts of the XACML enforcement infrastructure. Each compartment can be used to host service and applications. The shared compartment is used to host the XACML core components like the PAP, PDP, and PIP instance. The Policy Retrieval Point (PRP) is a spin-off of the PAP to highlight the separation between policy retrieval and policy administration. Furthermore, each compartment contains so-called Data Transformation Services (DTS) which apply transformations like

[5] http://www.sunfishproject.eu/

encryption, data masking, or anonymisation on transferred data. The crux is the PEP at the edge of each compartment, which guards the access originating in its compartment and coming to its compartment.

The PEP is implemented as a transparent gateway or proxy which inspects all the traffic and transparently evaluates policies. The policies are, as intended by XACML, defined based on attributes. Therefore, the PEP is required to extract attributes describing the data flow without forwarding the complete data to the PDP. If access is permitted by the PDP, the PEP transparently connects to the destination service, which again is intercepted by the PEP in the destination compartment.

Data usage control is not directly foreseen by XACML. With a clear enforcement model as defined in the SUNFISH project, it is clear how data flows through the system. Using this information, data usage control mechanisms are defined based on the obligation mechanism of XACML. Obligations enable policy creators to execute defined actions at the PEP. The extensions applied in the SUNFISH project enable to define obligations to apply data transformations on request and response data, supported by the DTS. The DTS abstracts tasks like key management for encryption and data masking, and takes this burden off the PEP and the application.

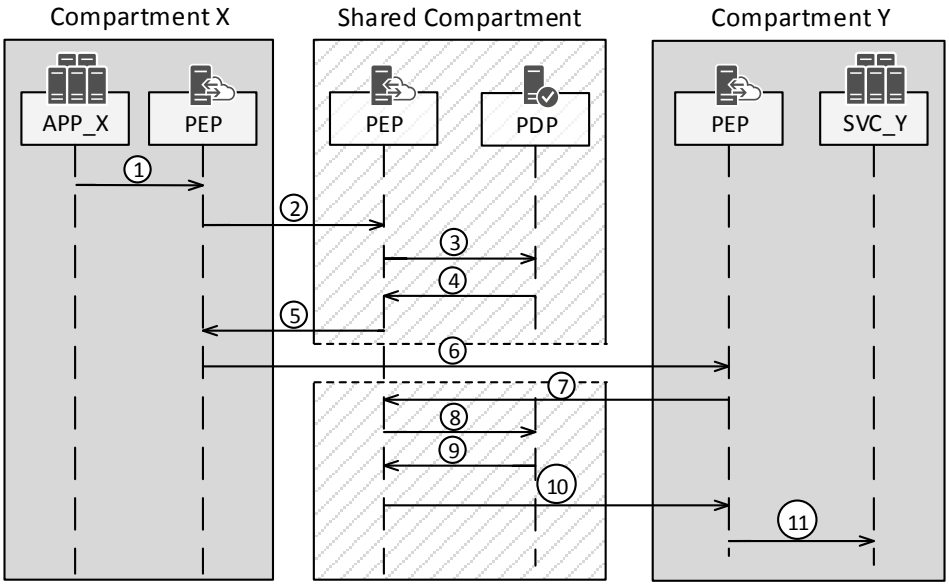


Figure 11.4: SUNFISH example interaction

Figure 11.4 illustrates a simplified interaction between an application and a service. The figure only shows the request phase, but the response is handled analogously. The application *APP_X* is deployed in *Compartment X*, whereas the service *SVC_Y* is deployed in *Compartment Y*. The *Shared Compartment* contains all the XACML core components. The figure only highlights the enforcement flow, flows like gathering additional attributes are omitted. The numbers in the figure correspond to the following process description:

(1) *APP_X* issues a request to *SVC_Y*. The request is intercepted by the PEP at the edge of the origin compartment.

(2 - 3) Based on the application-provided data, like target host, endpoint, payload, request type, user, origin compartment, target compartment, current compartment, and other meta-data, a XACML decision request is assembled and sent to the PDP. The decision request maps all discovered characteristics to XACML attributes identified by category and attribute identifiers.

(4 - 5) The PDP gathers relevant policies, resolves missing attributes and evaluates the decision request. The resulting decision response is sent back to the PEP in the origin compartment.

(6) The PEP enforces the provided obligations. In case of a negative PDP response the request is cancelled and an error is returned to *APP_X*, otherwise the request is forwarded to *SVC_Y*.

(7 - 10) The request again is intercepted by the PEP in the target compartment and actions analogous to steps (2 - 6) are executed.

(11) Finally the request is forwarded to *SVC_Y*.

Besides providing pure authorisation and blocking whole compartments from accessing a deployed service, this approach enables e.g. in step (5) to apply data transformations. Considering *SVC_Y* as an analytic service which only needs access to parts of the provided data, one could create policies which require the PEP in *Compartment X* to mask sensitive data before passing the compartment borders. This way, only the relevant data is transmitted to *Compartment Y* which decreases the risk of sensitive data disclosure.

If sensitive data is not allowed to be processed outside private cloud borders, this approach still enables operations on uncritical data with automatic data masking, encryption, or other data transformation methods. Furthermore, multiple private cloud providers could dedicate computing resources to a special compartment where high-security hurdles need to be overcome. This approach enables to federate multiple private clouds and still respect data security requirements of data owners.

11.3 Chapter Conclusions

In this chapter, the basics of the de-facto standard in data security policy languages, XACML, were introduced. XACML defines a hierarchical approach for developing data security policies and fosters reusability. Beside a policy language, the XACML specification also defines abstract approaches to policy evaluation and enforcement, the latter only being very vaguely. In fact, the specification requires each application or service entity to either explicitly interact with custom enforcement components or to include large parts of the enforcement logic. This is picked up by the H2020 SUNFISH project using the example of private cloud federations for the Public Sector. The top requirement is to also support legacy applications without requiring a complete re-implementation. This heavily inspired the development of the enforcement architecture. The result is a compartment-based approach, where access to services and application deployed inside a compartment is guarded by PEPs at the edge of each compartment. The duties of the PEPs were extended, compared to the XACML specification. The PEP not only acts as an external enforcement point but as an enforcement gateway with full control over the data flows. This enables the usage of private cloud federations, also for services operating on sensitive data. The author of this thesis was in charge of the SUNFISH work package, where the SUNFISH governance model was developed. While SUNFISH was a collaborative research project, he is the main author of the governance model. He derived the model based on the requirements of the specified use case, coordinated different partners and finally guided the development process. All components from the enforcement model were realised connected to multiple DTSs (data masking, anonymisation, and secure multiparty computation). Furthermore, three real-world use cases from different partners across Europe were transformed to use the SUNFISH model.

In this chapter, no obvious link to HMEC was provided, but an approach from a related domain is highlighted. The following chapter will show how XACML in general and the approaches developed in the SUNFISH project are related to HMEC and how they can be re-used to increase data security and privacy in HMEC use cases.

Chapter 12

Enable HMEC to Operate on Sensitive Data

Private users and especially corporations have strong requirements on the privacy and confidentiality of processed data. Existing offloading frameworks and solutions do not consider security issues at all and focus on implementation aspects. This chapter focuses on establishing means to enable usage of HMEC in scenarios where data security is among the top priorities. In Chapter 11 two user groups, private users and corporate user, were established. It is shown that both user groups have different preconditions. Therefore, user group dependent means are available to establish a security- and privacy-aware HMEC framework. In this chapter, the developed SUNFISH security model is translated to the use case of HMEC with the goal of fostering re-usability of already available infrastructures and identifying similarities with the SUNFISH defined enforcement infrastructure, but still making use of advantages introduced by HMEC frameworks. This chapter is based on the following publication [Reiter, 2017].

12.1 Enhancing the Enforcement Infrastructure

A typical HMEC use case for corporations could manifest as follows: A corporation is deploying an application with computational expensive parts to their employee's mobile devices. The application processes sensitive data which must not be exposed to third parties. One characteristic of this application might be a requirement for frequent interactions between the application and the computationally expensive part. A classic approach to realise this application includes providing a service in the corporation's private cloud, which can be accessed by corporate managed devices.

The designated goal is to achieve a system where defined HMEC means can be used as illustrated in Part III of this thesis. Still, the corporation or user remains in full control of its data and can explicitly allow or deny offloading operations based on factors like offloading destination, processed data, or current location of the user. This list of factors should be expandable to allow a variety of applications with different requirements.

Abstracting this typical use case reveals a similar structure as defined in the SUNFISH project. Each mobile device, a local computing unit, or cloud computing unit can be considered as a separate compartment. Compartments in the SUNFISH universe encapsulate services and applications with similar data security requirements. Accesses outside its compartment are protected by the PEP, which contacts the core policy framework for policy evaluation and enforces the received decisions. Converted to the HMEC universe, this means that each framework instance has a lightweight embedded PEP, communicating with the company controlled policy evaluation infrastructure as illustrated in Figure 12.1. The SUNFISH architecture is tailored to XACML as the de-facto standard in data security policy languages, but can easily be adapted to other policy languages.

The main policy related components in the shared compartment stay unchanged. A *HMEC Adapter* is added to translate between the HMEC required framework and the policy enforcement infrastructure

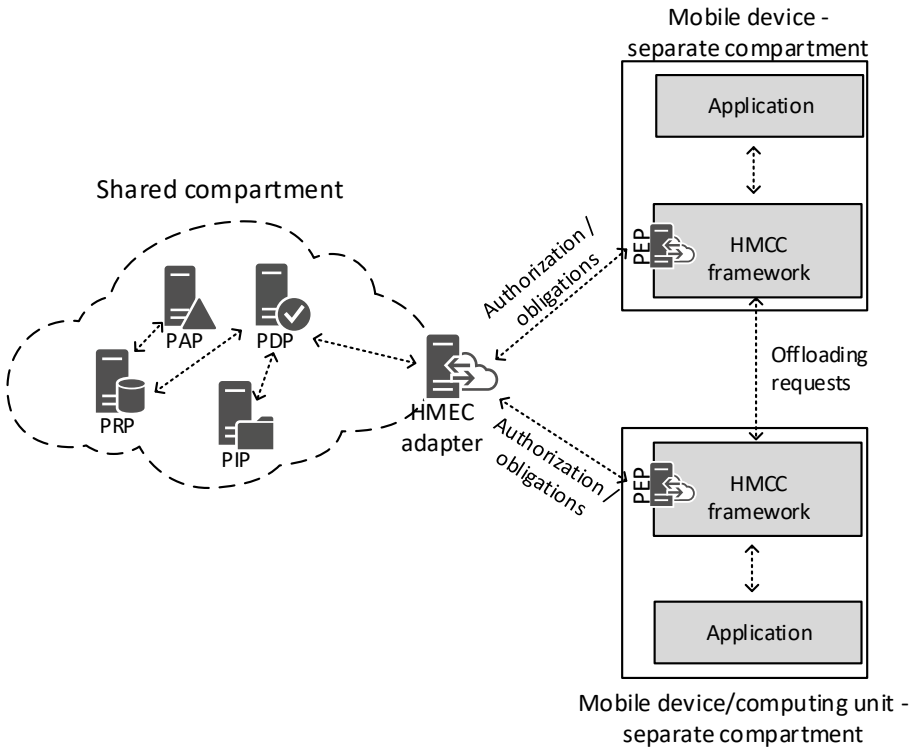


Figure 12.1: XACML integration into HMEC frameworks

provided format. Details about the requirements on the policy language, the *HMEC Adapter* and its protocols are provided in the following sections.

12.1.1 Requirements on the Policy Language and Evaluation Infrastructure

The XACML specification defines a very flexible data security policy language, suitable for different scenarios. This open specification requires guidance for specific use cases. In this section, requirements on the HMEC tailored policy language are derived from the high-level requirements as defined in Chapter 8.2: *lightweight*, *flexible*, and *interoperable*.

To make use of the full potentials of the HMEC approach and fulfil the requirements for a lightweight and flexible approach, it is required that mobile devices stay in complete control of the offloading operation and only use the XACML decision infrastructure infrequently to acquire allowed actions. As it is the case in the SUNFISH approach, the PEP issues a request to the policy decision engine and describes the processed data without transferring the whole datasets. Therefore, the first requirement can be defined as:

R 1 *Policies need a flexible way of specifying the targeted input data range.*

To not disturb the HMEC operation and unnecessarily mitigate the performance and energy saving gains, the actual offloading operation still remains in full control of the client device and adheres to the defined requirement on flexibility. Still, policies need to control, to a certain extend, which computing units are allowed to be used for a particular operation:

R 2 *Policies need to be able to specify nodes, eligible for the offloading operation in a general way. Concrete nodes available at runtime are not known.*

The policy evaluation infrastructure can be considered as a single-point-of-failure. Although this can be mitigated with distributed and fail-safe deployments, Part III of this thesis shows that (a) latency at end user devices is subject to high fluctuations due to unpredictable environmental conditions, and (b) the success of offloading frameworks highly depends on keeping latencies low. As the evaluation in Chapter 12.4 will show, policy evaluation is only subject to low latencies, but issuing decision requests for every offloading operation will render the system unusable. Furthermore, adherence to the defined requirement on flexibility needs to be respected. Therefore, the functionality of the framework needs to be maintained, also in case connectivity to the evaluation infrastructure is interrupted:

R 3 *Caching of policy decisions needs to be enabled. A sophisticated caching mechanism should enable a time-based caching, as well as a limited local re-evaluation of a policy decision.*

The policy evaluation platform is a critical infrastructure with full control of the offloading process. Therefore, assurance is required that communication only takes place with trusted endpoints:

R 4 *Policy decisions need to be cryptographically bound to a trusted policy decision infrastructure to prohibit forged decision responses.*

The requirement on interoperability is respected in two ways. Although this section is tailored to XACML as the de-facto standard in data security policy languages, the requirements and architecture can easily be adapted to other languages. XACML applies a general enforcement model which maps to other enforcement models. Interoperability on mobile devices' side is already provided by the defined operation environment from Chapter 9.1. The enforcement point running on the mobile device also needs to fit into this operation environment and therefore comes with interoperability properties as defined.

12.1.2 HMEC Tailored Policy Language

In general, the term *policy language* can be considered as an umbrella term defining languages for the following purposes:

- A language and format for defining policies,
- for issuing policy evaluation requests, and
- for receiving the policy evaluation results.

XACML provides a solution by defining a policy definition language, a decision request language and a decision response language. The goal of this chapter is to introduce the developed XACML extensions to fulfil the defined requirements from Section 12.1.1 and to propose optimisations targeted specifically at the HMEC use case. The proposed optimisations are fully integrated with the POWER implementation from Part III of this thesis. During research on this topic, the re-usability aspect was always kept in mind. Existing and already deployed policy enforcement infrastructures (primarily XACML infrastructures) need to be re-used. The following chapters will show (a) the fulfilment of the defined requirements by baseline XACML implementations and (b) further improvement and optimisation steps specifically targeted at the HMEC use case.

12.1.2.1 R 1 - Flexible Specification of Targeted Input Data

The fulfilment of **R 1** is already provided by the baseline XACML specification, and basically is the prime example of adding targets and conditions to XACML policies and rules. A basic example of specifying targets in XACML is provided in Listing 12.1. The target matches for all *source_address* parameters with the specified value. Various comparison functions are offered by the XACML specification, targeted at all different available data types. Still, evaluations on particular parameter can only

be applied if their value is provided to the policy decision infrastructure. Therefore, a tight collaboration between the application developers and policy creators is required. Technical aspects on the realisation are discussed in Chapter 12.3.

```
...
<Target>
 <AnyOf>
  <AllOf>
   <Match MatchId="...:function:string-equal">
    <AttributeDesignator
     Category="urn:power:1.0:parameters"
     AttributeId="source_address"
     DataType="http://www.w3.org/2001/XMLSchema#string"
     MustBePresent="false"/>
    <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string">
        10.27.152.153</AttributeValue>
   </Match>
  </AllOf>
 </AnyOf>
</Target>
...
```

Listing 12.1: XACML target matching on a specific attribute value

12.1.2.2 R 2 - Allowed Offloading Targets

With the fulfilment of **R 2**, controlling the allowed execution targets is possible. Generally, the computing units available at runtime are not known to the policy creators or can only be estimated. Therefore, policy creators are able to specify the allowed computing units in multiple ways. For the operation on uncritical data and processes, policy creators can enable offloading to all computing nodes, without evaluating their trustworthiness. For the operation on critical and sensitive data, policy creators can either directly specify allowed nodes (e.g. under control of trusted entities), or specify the requirement to pass attestation methods, as specified in Chapter 12.1.3. This requires a tight cooperation of the deployed applications and the policy evaluation infrastructure. In XACML this can logically and syntactically be mapped to the obligation mechanism. Listing 12.2 illustrates the simplest mapping of *NO_REQUIREMENT* to an obligation.

```
...
<ObligationExpression
 ObligationId="urn:power:1.0:obligation:execution-mode"
 FulfillOn="Permit">
 <AttributeAssignmentExpression
  AttributeId="urn:power:1.0:execution-mode">
  <AttributeValue
   DataType="http://www.w3.org/2001/XMLSchema#string">NO_REQUIREMENT</
    AttributeValue>
 </AttributeAssignmentExpression>
</ObligationExpression>
...
```

Listing 12.2: XACML obligation specifying no trust related requirements

Other valid values are *TRUSTED* and *SPECIFIC*. *TRUSTED* instructs the target application to perform an attestation of the trustworthiness of the current device state. An integration flow is illustrated in Chapter 12.1.3. *SPECIFIC* only allows execution on pre-defined computing nodes, specified as illustrated in Listing 12.3. Besides the execution mode, a set of allowed endpoints is added as a separate attribute,

also containing a link to their identity e.g. in the form of providing a certificate fingerprint used by this particular endpoint.

```
...
<ObligationExpression
 ObligationId="urn:power:1.0:obligation:execution-mode"
 FulfillOn="Permit">
 <AttributeAssignmentExpression
  AttributeId="urn:power:1.0:execution-mode">
  <AttributeValue
   DataType="http://www.w3.org/2001/XMLSchema#string">SPECIFIC</AttributeValue>
 </AttributeAssignmentExpression>
 <AttributeAssignmentExpression
  AttributeId="urn:power:1.0:node">
  <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:string-bag">
   <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string">
    {"endpoint": "client1", "identity": "[CERT-FINGERPRINT]"}
   </AttributeValue>
   <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string">
    {"endpoint": "client2", "identity": "[CERT-FINGERPRINT]"}
   </AttributeValue>
  </Apply>
 </AttributeAssignmentExpression>
</ObligationExpression>
...
```

Listing 12.3: XACML obligation specifying specific nodes to be used for execution

The previously described mechanisms enable to define particular nodes where offloading is allowed. Still, in most environments, many of the surrounding devices will have an undetermined trust state, hence will not be eligible for offloading sensitive tasks. This fact basically cannot be changed without considering the anatomy of tasks. Some tasks may actually operate on sensitive data, whereas others may just accept sensitive data embedded in data structures, but may never use it. Taking a simple illustrative task which sums up the *amount* field of the data structure as shown in Listing 12.4. The credit-card information and last name can be considered as sensitive information, but actually, is never used by the task, still only offloading to trusted computing nodes is allowed.

```
{
 "purchases" : [
  { "firstname" : "John",
    "lastname"  : "Doe",
    "creditcard" : "1234-5678-1234-5678",
    "amount" : 500.00 }, ...
 ]
}
```

Listing 12.4: Example data structure containing sensitive data

This is tackled by introducing client-side data transformation obligations. These obligations require a preprocessing of selected arguments, using a specified data transformation mechanism on the client side before the offloading process is initiated. Listing 12.5 illustrates a regular expression-based masking obligation. The obligation instructs the client application to apply asterisks on argument *arg1*, for matching parts of the specified regular expression. In the listed example, the regular expression matches all, but the last group of standard credit card numbers. Various obligations can be defined matching different data structures and protect sensitive data. After masking, the data structure should not contain any sensitive data and can be offloaded to untrusted computing units, provided that the offloaded task does not require access to the masked sensitive data.

```
...
<ObligationExpression
 ObligationId="urn:power:1.0:client-obligation:regex-asterisk-masking"
 FulfillOn="Permit">
 <AttributeAssignmentExpression
  AttributeId="urn:power:1.0:client-obligation:argument:argument-name">
  <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string">
   arg1</AttributeValue>
 </AttributeAssignmentExpression>
 <AttributeAssignmentExpression
  AttributeId="urn:power:1.0:client-obligation:argument:regex">
  <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string">
   (\d[ -]*?){9,12}
  </AttributeValue>
 </AttributeAssignmentExpression>
</ObligationExpression>
...
```

Listing 12.5: XACML obligation specifying a masking operation

12.1.2.3 R 3 - Evaluation Result Caching

The methods and techniques presented in this work propose a decentralisation of components using HMEC approaches. Including centralised policy evaluation structures, vital for the operation of the solution, is again a step towards centralisation and has negative impacts on the overall performance of the framework as illustrated in Chapter 12.4. Therefore, minimisation of the impacts on performance and availability is required. This is achieved by keeping the dependencies on the centralised policy evaluation infrastructure low. Policy evaluation decisions are stored and re-evaluated locally. Only in case no valid cached policy decisions are available, the policy evaluation infrastructure is contacted. The caching of policy evaluation decisions can be controlled by the policy creators via two aspects mapped to obligations. They can also be combined within single policies:

- *Time-based* validity constraints as illustrated in Listing 12.6. The illustrated obligation allows a client side usage of decision responses for one hour after being issued.

```
...
<ObligationExpression
 ObligationId="urn:power:1.0:obligation:validity"
 FulfillOn="Permit">
 <AttributeAssignmentExpression
  AttributeId="urn:power:1.0:validity">
  <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#integer">3600</
      AttributeValue>
 </AttributeAssignmentExpression>
</ObligationExpression>
...
```

Listing 12.6: XACML obligation specifying a time-based validity constraint

- *Content-based* validity constraints as illustrated in Listing 12.7 enable the specification of input data ranges, where the policy decision stays valid. Parameters are checked on the client-side if they match in the defined constraints and allow reusing the acquired decision responses. The constraints are specified using *JSON Predicates*[1]. Parameters and context data required for the execution are mapped to a JSON structure and are matched against the defined constraints. JSON predicates

[1] https://tools.ietf.org/id/draft-snell-json-test-01.html

then enable to build logical combinations of different match operators. The illustrated example references the value of parameter *arg1* with */parameter-values/arg1* and checks if the parameter has the value *value1*. Various predicates, e.g. boolean operations, operations on numbers, or matching against regular expressions are available and can easily be extended to match complex structures.

```
...
<ObligationExpression
 ObligationId="urn:power:1.0:obligation:validity"
 FulfillOn="Permit">
 <AttributeAssignmentExpression
  AttributeId="urn:power:1.0:constraint">
  <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string">
  {
  "op" : "and",
  "apply" : [
   {
    "op" : "test",
    "path" : "/parameter-values/arg1",
    "value" : "value1"
   },
   {
    "op" : "less",
    "path" : "/parameter-values/arg2",
    "value" : 5
   }]
  }
  </AttributeValue>
 </AttributeAssignmentExpression>
</ObligationExpression>
...
```

Listing 12.7: XACML obligation specifying a content-based validity constraint

These measures enable caching and reusing of acquired policy decisions on the client-side, without contacting the policy evaluation infrastructure, therefore reducing the dependencies on centralised infrastructure and decreasing the latency of each single offloading request. The impacts of these measures are shown in Chapter 12.4.

12.1.2.4 HMEC Optimisations

XACML and XML, in general, are very verbose languages, serving all possible use cases. Additionally, XML requires a lot of parsing effort compared to lightweight approaches like JavaScript Object Notation (JSON). JSON improves human readability of plain data but decreases the expressiveness. As highlighted by Lin et al. [2012] as well as by Nurseitov et al. [2009], parsing JSON content is faster compared to parsing XML and consumes less resources. Therefore, for targeted use cases with clear boundaries and where strict timely constraints apply, it is more efficient to define a targeted language using JSON. Still, existing technologies and already deployed solutions should remain usable and policy creators should still be able to use their familiar tools. Therefore, the *HMEC Adapter* component, as illustrated in Figure 12.1, was introduced. This component acts as a translation gateway between the verbose XACML decision request/response language and the lightweight JSON based language as introduced in this chapter.

An appendix to the XACML specification already defines a JSON profile [OASIS, 2014] for the interactions between PEP and PDP. Besides the increased parsing effort, XML structures have an increased length compared to JSON structures according to Sumaray and Makki [2012]. These observations do

not hold for the XACML JSON profile because XACML is targeted at XML and makes heavy use of features like XML namespaces [Bray et al., 2009] and XPath [Robie, Dyck, and Spiegel, 2017]. To maintain the functionality, a translation to the corresponding format is required. The problem is illustrated in the following two listings 12.8 and 12.9, showing the transmission of an attribute from the PEP to the PDP embedded in a decision request (not shown). Due to the various utilised namespaces and XPath selectors, the attribute representation can only be decreased from 404 bytes to 375 bytes in size, excluding not required whitespaces.

```
<Attribute xmlns="urn:oasis:names:tc:xacml:3.0:core:schema:wd-17"
 AttributeId="urn:oasis:names:tc:xacml:3.0:content-selector">
 <AttributeValue xmlns:md="urn:example:med:schemas:record"
 XPathCategory="urn:oasis:names:tc:xacml:3.0:attribute-category:resource"
 DataType="urn:oasis:names:tc:xacml:3.0:data-type:xpathExpression">md:record/md:
    patient/md:patientDoB</AttributeValue>
</Attribute>
```

Listing 12.8: XACML attribute represenation in XML [OASIS, 2014]

```
{
 "Attribute": {
  "AttributeId": "urn:oasis:names:tc:xacml:3.0:content-selector",
  "DataType": "xpathExpression",
  "Value": {
   "XPathCategory":
     "urn:oasis:names:tc:xacml:3.0:attribute-category:resource",
   "Namespaces": [
    {"Namespace": "urn:oasis:names:tc:xacml:3.0:core:schema:wd-17"},
    {"Prefix": "md",
     "Namespace": "urn:example:med:schemas:record"}
   ],
   "XPath": "md:record/md:patient/md:patientDoB"
}}}
```

Listing 12.9: XACML attribute representation in JSON [OASIS, 2014]

Therefore, to minimise the transported data sizes, a use case specific language is required. Large parts of the request/response language can be introduced for general use cases. Still, adaptations are required for concrete target use cases. The remainder of this chapter introduces a language targeted at the framework as proposed in Part III of this thesis. Use case specific parts are highlighted. The following Listing 12.10 illustrates a simple decision request example generated by client applications on mobile devices. The bold parts are specific to the selected use case and require adaptation. The decision request needs to describe an offloaded task as specific as required, but should not contain unnecessary data. This results in a tight cooperation between implementation aspects as highlighted in Chapter 12.3 and the concrete optimised decision request/response language. A minified version (unnecessary whitespaces removed) of an exemplary JSON-based decision request has a length of 271 bytes, whereas the minified version of the corresponding XACML decision request in XML format (not shown) has a total length of 1755 bytes, which corresponds to nearly 85% savings in size.

The request language can be separated in three major parts, illustrated by the colours blue, red, and green:

- Fields highlighted in *blue* are related to uniquely identifying single requests. An ID is randomly generated on each request, and matches the ID in the corresponding response.
- Fields highlighted in *red* identify the task to be executed. In the context of the introduced HMEC framework a task is uniquely identified by its package (or application-) name and version, as well

as a method name in the application scope. This enables policy creators to clearly separate policies by application, without unwanted interference between policies of different applications.

- Fields highlighted in *green* specify the context of the planned task execution, like parameters and other arguments to the task. These parameters can be used by policy creators to further constrain the applicability of policies and control the caching mechanisms.

```
{
 "id": "[request-id]",
 "package-name": "[package-name]",
 "application-version": "[application-version]",
 "method-name": "[method-name]",
 "cookie-values": {
  "cookie1":"cookie value 1",
  "cookie2":"cookie value 2"
 },
 "parameter-values": {
  "arg1" : "value1",
  "arg3" : [1, {"complex": "value"}]
 }
}
```

Listing 12.10: Use case specific decision request language

In contrast to the decision request, the decision response, as illustrated in Listing 12.11, does not contain use case targeted fields and is kept on a generic level. Still, a similar separation in three major parts can be applied:

- Fields highlighted in *blue* are again related to identifying the response and build a connection to the corresponding request. The contained ID matches the request ID.
- Fields highlighted in *red* specify nodes allowed to be used for the offloading operation, as well as obligations to be fulfilled. This is a lightweight translation of the means defined in Section 12.1.2.2 to the optimised format. The allowed nodes and obligations are tightly connected. Obligations can be flagged as mandatory to only allow offloading to the specified computing units given the obligation has been fulfilled.
- Fields highlighted in *green* control the caching and re-usability of the decision response and are a lightweight translation of the means defined in Section 12.1.2.3 to the optimised format.

```
{
 "id": "[request-id]",
 "request": "[link_to_request]",
 "execution-mode": "specific",
 "nodes": [{
  "endpoint": "client1",
  "identity": "[identity e.g. certificate fingerprint]"},
  {
  "endpoint": "client2",
  "identity": "[identity e.g. certificate fingerprint]"}],
   "obligations": [
   {
    "transformation": "regex-masking",
    "required": true,
    "parameters": {
    "regex": "(\\d[ -]*?){9,12}",
    "argument-name": "arg1"
```

```
  }}]
 "validity": {
  "current-time": "2017-01-27T15:42:44+0100",
  "valid-for": 3600,
  "constraints": [
  {"op" : "and", "apply":[
   {"op":"test", "path" : "/parameter-values/arg1", "value" : "value1"},
   {"op":"less", "path" : "/parameter-values/arg2","value" : 5}]}"
 ]}
 }
```

Listing 12.11: Use case specific decision response language

These introduced techniques allow the most efficient transport of task descriptions, for the purpose of determining the allowed offloading targets to a corporate/user controlled policy decision environment and enable a broad re-usability of acquired decisions. Additionally, the HMEC adapter can augment incoming request with additional context information to enable a context-aware definition of policies:

- *Location data:* Based on the incoming IP information, approximate geographic location information can be added.
- *VPN data:* Information regarding the VPN connectivity can be added and can even be cross-checked with available VPN gateways.
- *On premise:* Add information if devices are located on-site and e.g. connected to a local wireless network.

Based on this information, policies can be further detailed to enable a more efficient usage of available computing units and a seamless user experience.

12.1.2.5 R 4 - Cryptographic Binding to Trusted Policy Decision Infrastructures

Besides transport security aspects, policy decision environments require a cryptographic binding of the issued decisions to the trusted policy decision infrastructure. The XACML specification refers to using XML signatures [Bartel et al., 2008] for this purpose. This, of course, is no longer a valid technology for the optimised JSON-based format. A lightweight signature solution for JSON exists, called JWT [Jones, Bradler, and Sakimura, 2015]. The compact JWT representation is an URL safe header, payload, and signature representation in the following format:

```
<header data Base64URL encoded>.<payload data Base64URL encoded>.<signature Base64
    URL encoded>
```

Listing 12.12: JWT structure

The decoded header data represent a JSON structure specifying the used algorithm, the actually signed content, and the signature according to the defined algorithms.

To establish a cryptographic binding between the policy decision environment and the application executed on a mobile device, the mobile application is equipped with the public key of the enforcement infrastructure at the time of enrolment. The HMEC adapter wraps the decision response in a JWT structure and attaches a signature, as illustrated in Listing 12.13. This signature needs to be checked by the client application. Besides the cryptographic binding, the response is bound to the corresponding request by (a) including the unique request ID for replay protection and (b) enabling to establish a link to the concrete origin request in the *request* field. This *request* field contains the SHA-256 hash of the originally issued request payload, to ensure the integrity of the evaluated request.

```
eyJhbGciOiJSUzI1NiJ9.eyJpZCI6[...]OltdfQ.X5e4epNjJbw[...]w8OcIe9-WlTr_aZU
```

Listing 12.13: Illustration of the signed JWT response

These means fulfil the requirement **R 4** and protect client applications from unauthorised modifications of the issued request, as well as provide a cryptographic binding to the policy decision environment. This binding then enables to delegate the offloading decision process.

12.1.3 Device Attestation Methods

This chapter tackles the problem of determining if a particular computing unit can be considered as trusted and envisions a deep integration with the means as described in Chapter 12.1.2.2. Device or platform attestation is key for offloaded tasks operating on sensitive data. Data controllers need assurance that the executing platform is trusted and sensitive data is not leaked to untrusted third parties. This can be achieved using organisational or technical measures. The described device attestation mechanisms are technical measures to ensure a trusted state of a device, platform, or more generally, of the environment. Effectively, device attestation methods are enablers for computational offloading involving sensitive data. Without considering a concrete technological solution, two integration models can be developed. These models differ in terms of where the assessment takes place:

- *Policy Decision Environment:* The policy decision environment anyhow acts as a trusted entity in the system, an enriched environment might even be part of the P2P communication network, thus can also directly contact single nodes. Performing the attestation at the policy decision environment increases the efficiency because attestations only need to be performed once, regardless which device is requesting the offloading operation. This includes that mobile devices need to suggest computing units in their proximity for attestation. The allowed nodes are then transported using the *SPECIFIC* identifier to the mobile device.

- *Mobile Device:* Policy decision environments can also delegate the attestation process to the mobile device, which is necessary for environments where no stable connection to the policy decision environment is available, or they are not capable of performing the assessment. Overall, this results in more assessments compared to the prior approach but is a further step towards a decentralised approach.

Both methods are considered by the developed HMEC decision request/response language, either by transferring concrete computing unit identifiers and their used certificates (*SPECIFIC* computing units), or by just referring to *TRUSTED* computing units, and delegating the assessment to the mobile devices. In the following, approaches are considered enabling a remote mobile device and computing unit attestation regarding their trust state.

For the POWER framework implementation in Chapter 9.2 two communication technologies were used. The implementation using the decentralised P2P framework provides a full-featured certificate-based attestation implementation. For the other attestation methods as described in the following chapters, the focus was not put on the implementation aspect but on integration strategies with the certificate-based approach.

12.1.3.1 MDM- and Certificate-based Solutions

Mobile Device Management (MDM) is a way of centrally managing mobile devices and targets at protecting the data on the devices. Rhee et al. [2013] elaborate a threat model for MDM systems and identified possible attacks. Figure 12.2 shows a simplified architecture, as used by Rhee et al. [2013] for their work. Basically, a MDM solution consists of four essential components:

- The *MDM agent* is an application executed on the mobile device. It collects data relevant for assessing the state of a mobile device and sends it to the *MDM server* for analysis. Furthermore, it executes actions requested by the *MDM server*. Possible actions include enterprise application management, remotely wipe the device, update the system configuration like adding certificates, and disable or enable certain features of the device like Bluetooth, GPS, or camera.
- The *MDM server* is the heart of a MDM solution. It manages all registered devices and their associated configuration. Furthermore, it assesses, based on data received from the *MDM agent*, if a mobile device is in a legitimate state. The assessment may also be shared between the *MDM agent* and the *MDM server* but this can be considered as a detailed implementation aspect out of scope of this paragraph.
- The *MDM management console* is an administrator-focused application for configuration, where all the possible actions can be triggered.
- The *MDM relay server* acts as a gateway between servers deployed in the internal enterprise environment and mobile devices in the field, connected via the Internet.

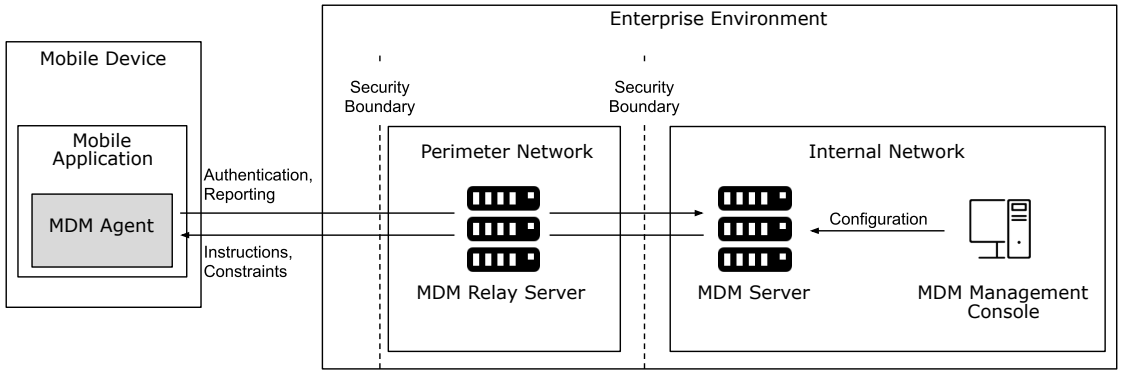


Figure 12.2: MDM overview, based on the work of Rhee et al. [2013]

The description above makes it clear that MDM solutions are only suited for corporations including Bring Your Own Device (BYOD) scenarios. HMEC frameworks can profit from MDM solutions by delegating the trustworthiness assessment. This only works if offloading takes place on corporation controlled devices. This definitely renders a valid use case in corporate environments with strict data protection requirements. Integrating multiple corporations with different MDM solutions still is a complex task. Therefore, an integration strategy is presented here, which boils down MDM solutions to certificate-based solutions. The integration of certificate-based solutions, in turn, is straight forward. The mobile device or computing unit used for offloading authenticates at the user's mobile device using its certificate issued by a potentially trusted entity. Depending on the integration model, either the policy evaluation infrastructure or the mobile device performs a certificate validation and checks if the trustworthiness of the certificate issuer matches the required trust level.

This leaves the open point of how devices get their trusted certificates. Analysing the situation reveals a three-staged process involving the MDM solution and continuous monitoring as illustrated in Figure 12.3. In the initialisation phase, the MDM server is equipped with the policies in force, the MDM agent is installed on devices and is coupled with the MDM server. The policies express requirements on (depending on the capabilities or the concrete MDM solution):

- lock screen security like the complexity of passwords or allowed types of lock-screen passcodes,
- full device encryption,

- allowed features of the device like bluetooth, camera, or voice assistants,
- allowed software
- allowed operating systems and allowed operating system versions.

The reporting phase is a continuous process on the mobile device where the adherence to defined policies is validated. This process is carried out between the MDM agent and the MDM server. The MDM agent reports configuration data and device statistics to the MDM server, which then evaluates the conformance to defined policies. Once a device is compliant, the MDM server provisions a certificate to be used for offloading operations. Still, the reporting phase continuous. If a non-conformance is detected, the device's certificate needs to be revoked and removed from the device.

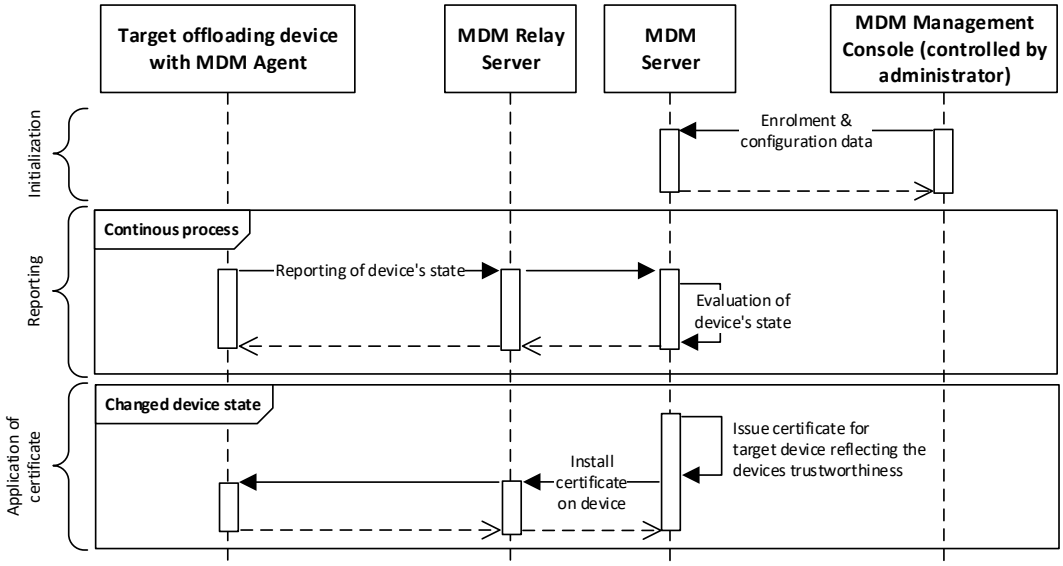


Figure 12.3: HMEC MDM integration

This way, corporations with different MDM solutions can establish trust relationships, and cooperate regarding resource usage without diving into the very details and particularities of all different MDM solutions.

12.1.3.2 Google Safetynet-based Solution

Mobile devices offer good isolation between applications. If security mechanisms of recent mobile devices are used, applications can protect their data in a way that no other entities can access the data: applications do not have access to other application's data and can use device-provided secure elements to store sensitive data or perform critical cryptographic operations. The SafetyNet API[2] is a service included in Google's Play services, thus available on all recent Android releases which enables the attestation of a device. A positive result strongly indicates that no one tampered with the device, the device is not rooted and application isolation can be guaranteed.

The SafetyNet attestation approach relies on two components: The closed source *snet* service is constantly running in the background on the mobile device. It periodically collects *relevant* data and reports this data to Google servers. There is no confirmed public information available which data Google considers as relevant, but based on this data Google performs server-side analysis and can tell if the device currently is in a non-tampered state. The *snet* service is not part of any APK file, it directly

[2]https://www.cigital.com/blog/using-safetynet-api/

connects to Google servers and downloads a binary package if an update is required. This is a very flexible approach and enables frequent updates, without going through the Google Play Store. The second part is the attestation API, accessible via the Google Play Services. It enables to request an attestation statement, which represents a signed statement about the current state of the device based on the gathered data.

The attestation result is a JWT signed JSON structure and contains the following information:

- A unique nonce generated on the client device as sent in the attestation request,
- the APK package name of the requesting package,
- hash of the certificate used for signing the requesting application,
- hash of the requesting APK, and
- an indicator if the device passed the tamper check.

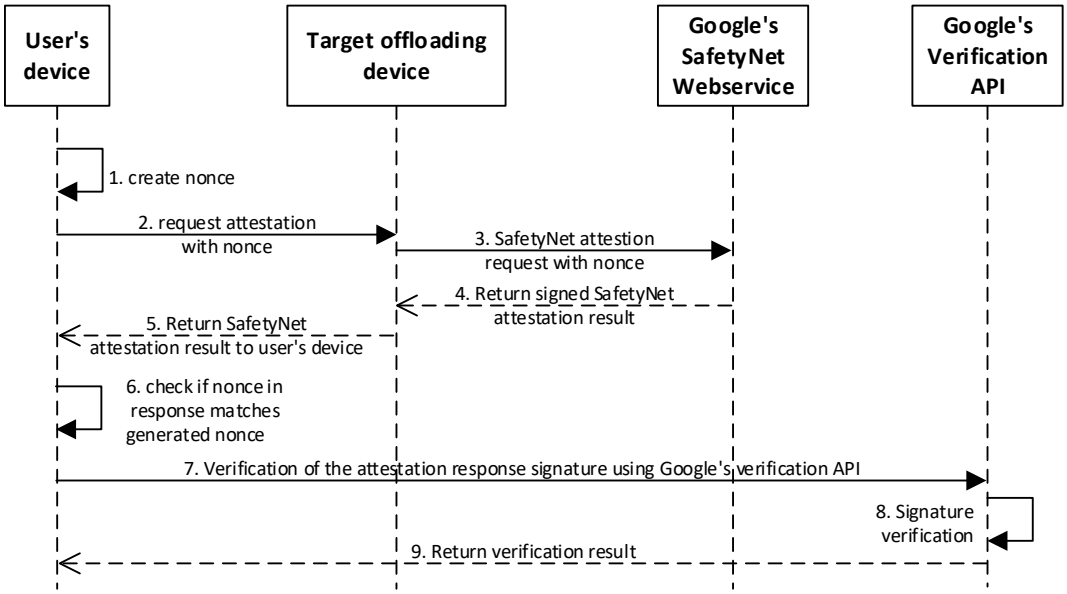


Figure 12.4: HMEC SafetyNet integration

In fact, this approach is very similar to the MDM based approach, without the capability to customise the policies. Therefore, to make use of this approach, a direct integration is necessary as illustrated in Figure 12.4 involving the following steps:

1. The user's device creates a random nonce, so it can map the received attestation response to the particular request clearly.
2. The target offloading device is contacted through a provided communication channel.
3. The target device issues a SafetyNet attestation request to the attestation web service, including the nonce as generated by the user's device.
4. The generated attestation response is forwarded to the user's device.
5. The user's device checks if the nonce in the response matches the created nonce from the first step. If they do not match, the attestation process fails.

6. To verify the validity of the signature of the attestation response, a verification request is sent to Google's verification API.

7. Based on the verification result and the status of the attestation, the user's device decides if the target device meets the requirements.

This approach is the counterpart of MDM-based solutions for private users. A major issue of both technologies, the MDM and Safetynet approach, is their reliance on software-only methods. Therefore, security guarantees are not valid for rooted devices. On a rooted device, the MDM agent and the *snet* service can be tricked into reporting faulty values, resulting in a positive attestation result.

12.1.3.3 Intel SGX-based Solution

Intel introduced the Software Guard Extensions (SGX) [Anati et al., 2013] recently, which enable the creation and execution of protected software containers, also called enclaves. It is out of the scope of this thesis to go into the very details of this technology, but a complete explanation is provided by Costan and Devadas [2016]. Enclaves are small parts of applications, for example, single functions processing sensitive data. The crux with Intel's SGX is that the operating system, hypervisors, or any other software running on the system are unable to influence or inspect the execution of the enclave. Enclaves are protected by hardware-enforced access control policies, and the trusted computing base is reduced to the CPU, its hardware logic, microcode and a pre-provisioned SGX-enabled software component used for attestation. This represents a tremendous improvement compared to today's situation given SGX' modularity. Intel added special CPU instructions to create, load, and measure secure enclaves to ensure their integrity and enable the attestation of provisioned enclaves. Attestation is the process of ensuring that enclaves have been deployed correctly and are not modified. Furthermore, a trusted pre-provisioned enclave is added to enable remote attestation of the initial state of a provisioned enclave. Intel SGX makes a clear distinction between intra-enclave attestation and inter-platform or remote attestation. Intra-enclave attestation is based on symmetric cryptography and can only be verified on the same platform, whereas remote attestation involves public-key cryptography.

A typical remote attestation flow for SGX-enabled applications is illustrated in Figure 12.5. Here, a high-level overview of the procedure is provided, different attestation methods can be implemented on top of Intel's SGX. A SGX-enabled application is always separated into a trusted application, to be deployed in an enclave, and an untrusted application which launches the enclave executed on an untrusted operating system. The remote client is an entity with the intention to securely transport sensitive data to the application running in the enclave. The quoting enclave is a pre-provisioned enclave with access to signature credentials, verifiable by third party verification services. The goal of this remote attestation procedure is (a) to be sure that an unmodified application version is running in the enclave, and (b) that the remote client has a secured transport channel to the application running in the enclave. The following steps comprise a typical SGX-enabled attestation flow:

1. An untrusted application is executed on an untrusted operating system. It initialises and starts an SGX enclave using the added SGX CPU instructions. This also calculates a cryptographic measurement of the enclave after initialisation for later reference.

2. An external client wants to ensure that the enclave was initialised correctly and issues an attestation request to the untrusted application.

3. The untrusted application requests an intra-enclave attestation.

4. The intra-enclave attestation contains, besides the initial state of the application, user defined data to establish a secure communication channel. Therefore, an ephemeral public-private key pair is generated.

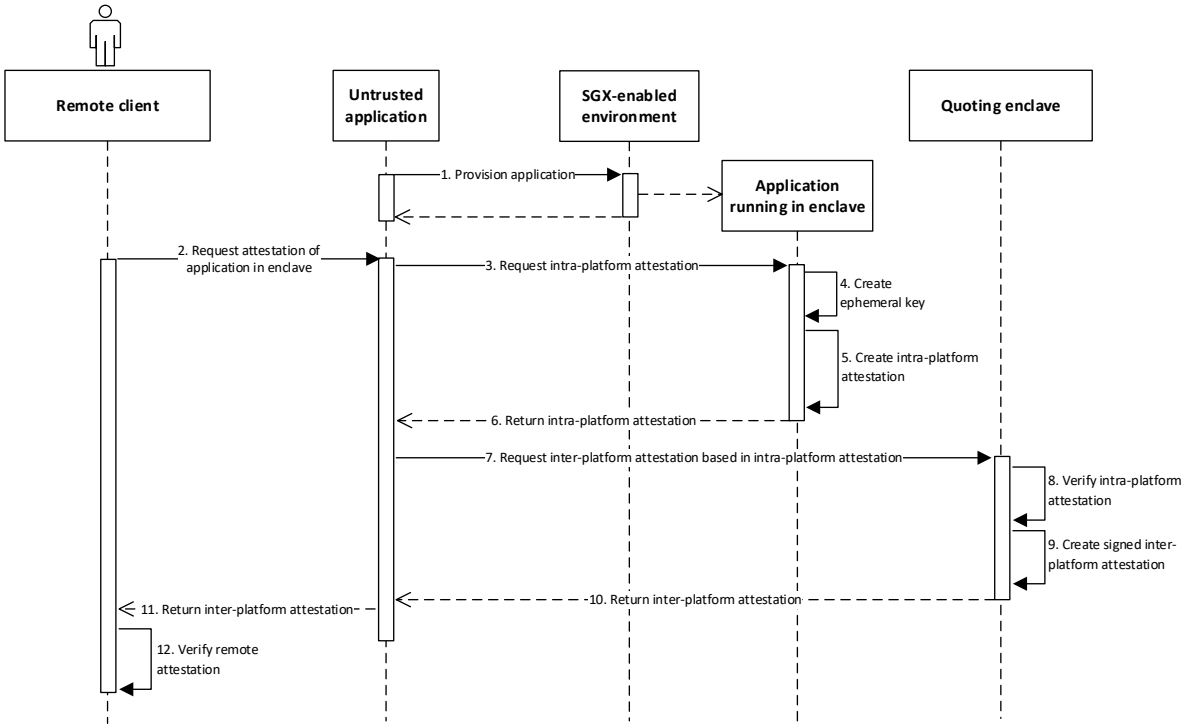


Figure 12.5: Typical SGX remote attestation flow

5. The attestation is created using the SGX *EREPORT* instruction, containing the initialisation state of the enclave and the provided user data. The attestation is cryptographically protected using a Message Authentication Code (MAC) with a key only accessible to the *EREPORT* instruction and the trusted quoting enclave.

6. The cryptographically protected attestation is returned to the untrusted application.

7. Next, the untrusted application requests an inter-platform attestation at the pre-provisioned quoting enclave.

8. The quoting enclave also has access to the MAC key and verifies the attestation result.

9. Once the verification has completed, the MAC is replaced with a signature created using a device specific private asymmetric key.

10. The enhanced attestation result is returned to the untrusted application.

11. The untrusted application returns the attestation to the calling remote client.

12. The remote client now needs to verify the attestation and used key using a third party service.

After the attestation procedure completes, the remote client can use the ephemeral public key, embedded in the attestation to securely communicate with the application running in the enclave. Furthermore, it can be sure, as long as the CPU manufacturer is trusted, that the application running in the enclave has not been modified and that no one can tamper this application.

For the integration of Intel SGX in HMEC frameworks, the application running in the enclave is a full-fledged computing node accepting offloading operations, and the untrusted application acts as a bootstrap application. This approach is currently hindered by several limitations of Intel's SGX in its current release like dynamic memory allocation, direct IO access, and only limited exception handling.

Nevertheless, Baumann, Peinado, and Hunt [2014] introduce a system called *Haven*. With Haven, they achieved to run legacy applications like Microsoft SQL Server and the Apache Webserver in an enclave. They are running a thin operating system layer in each enclave where the unmodified applications operate on. However, they reached the limits of SGX and proposed changes for the coming iterations of the specification. Their implementation is currently not available, as it requires source code level access to the Windows operating system kernel. Therefore, at the moment, this approach remains in a theoretical state. For the concrete integration, two approaches are conceivable. Each device could perform the remote attestation flow and verify retrieved attestation results on its own. This could result in a massive load of attestation requests. A more realistic integration model is to, again, boil down the application of SGX enabled computing units to the certificate-based approach. In this case, a trusted third party is required to perform all the remote attestations of SGX enabled computing units. Once the attestation is completed successfully, a certificate is issued and securely sent to the computing unit running in the enclave. This certificate is then used to authenticate against devices requesting offloading operations.

12.2 Trust Model

In this chapter, the trust model is introduced with its different trust states and the possible transitions between the different states. The trust transitions refer to the assigned trust levels of each component and their interactions with other components to determine their trust level. Through the interaction with other components, and with attestation mechanisms and protocols in place, components with an undetermined trust level can be evaluated. The source for the development of this trust model are the assessment methods from Chapter 12.1.3.1, Chapter 12.1.3.2, and Chapter 12.1.3.3. All described attestation methods, except the Safetynet approach, can be boiled down to using certificates. This is important concerning practical application of these methods. Therefore, to get a complete view of the trust mode and trust propagation, two trust assessment methods are considered, the SafetyNet approach and a certificate-based approach, as they are fundamentally different. Evaluating these two methods provides the full spectrum of assessment methods.

The trust model of this work is illustrated in Figure 12.6. It consists of three different trust levels:

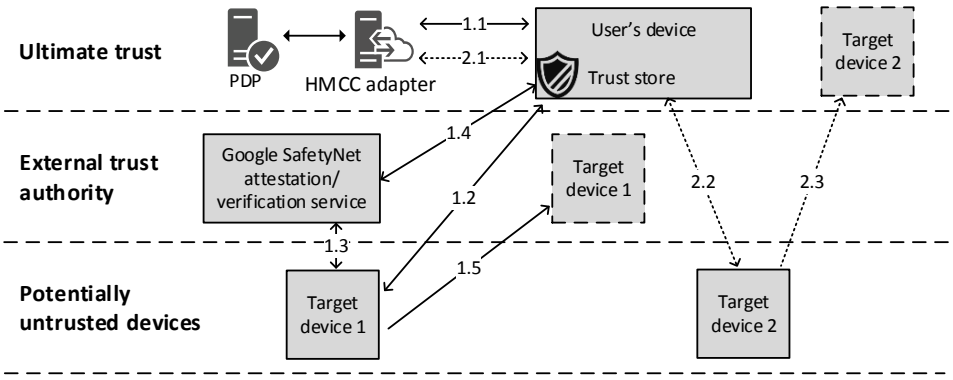


Figure 12.6: Trust model

- The *ultimate trust* level is assigned to the core policy decision components and the user's device, which typically is under corporation's control. In BYOD scenarios the corporation needs to assure that devices are under MDM control. These components act as trust anchors in the system. The trust level of other components is derived from their decisions also including delegated decisions as it is the case for MDM based solutions.

- The *external trust authority* level is assigned to approaches like the SafetyNet services. Other

related services can equally be assigned at this level or at higher levels. Depending on the corporation's policies, the external trust level might be equal to the ultimate trust level.

- The *potentially untrusted devices* level contains all other devices which might be untrusted, or might be in another trust level after assertion. This level also contains devices not under control of the corporation or tampered devices.

A more fine-grained classification can always be introduced at the data owner by creating policies accordingly. Still, these three levels represent a natural trust level selection. The policy evaluation environment is in full control of evaluating the data security policies and needs to be fully trusted. The same applies to the user's device. In HMEC scenarios, the application on the user's device is usually equipped with the complete application logic. If the purpose of this application is to process sensitive data, the user's device also needs to be fully trusted. For other devices, policy creators can make more fine-grained decisions to offload certain parts only to specific devices, as illustrated in Section 12.1.2.2. This also enables a distinction between different external trust authorities like Safetynet and different MDM solutions. Still, for evaluation and highlighting the trust propagation, it is adequate to combine them on a single level for external trust authorities. The remaining group of devices and computing units are those, where the evaluation results in a negative outcome, and the devices which have not been evaluated yet. Still, as long as they are not evaluated, they are potentially untrusted and can only be used for uncritical operations, or not at all.

Figure 12.6 contains the relevant interactions of the two described trust assessment use-cases indicated by **1.x** (trust assessment using the SafetyNet approach) and **2.x** (trust assessment using certificates). The following listings describe the necessary steps to determine the trust state of computing units:

(1.1) The user's device connects to the HMEC adapter. The HMEC adapter authenticates using HTTPS (or similar technologies), and the user's device verifies if the host is trusted by verifying the HMEC adapter's certificate against its trust store. At this point the application on the user's device can rely on the confidentiality, integrity and authenticity of the transmission channel to the terminating endpoint. Policy evaluation components might be protected by e.g. firewalls terminating the secured connection. Therefore, all policy decisions are explicitly signed by the HMEC adapter.

The application on the user's device then acquires a HMEC adapter signed decision result. The signature is checked against an application-embedded set of certificates on the user's device. A positive validation result ensures that the device acquired a legit decision response.

(1.2) It is assumed that the acquired results indicate that a *TRUSTED* device is required for offloading, which can be assessed using the Safetynet attestation service. The user's device finds *Target device 1* with Safetynet capabilities in its surrounding, and sends an attestation request including a fresh random nonce.

(1.3) The request is received by the currently untrusted device and is further dispatched to the attestation service.

(1.4) The attestation takes place as described in Section 12.1.3.2 and the issuing device receives the result.

(1.5) Supposed a positive attestation result was received, it is up to the data owner to define if Safetynet is considered as fully trusted or may only be used for certain operations. In any case, the trust level of *Target device 1* is increased to the level of Safetynet.

The described trust assessment workflow follows the decentralised paradigm adhered in this work. This gives the advantage of being independent of centralised infrastructures in large parts but also requires to potentially perform massive amounts of attestations. Actually, the same procedure can also be performed

by the policy evaluation infrastructure, which adds more dependence on centralised infrastructures, but definitely enables reusing previously acquired attestation results across multiple devices.

For the trust assessment based on certificates, a similar trust propagation workflow can be drawn:

(2.1) The first step is analogous to the first use case. In this scenario the policy creators have two distinct possibilities: (a) the policy enforcement infrastructure already provides a pre selected list of possible computing units, or (b) the result allows the usage of *TRUSTED* nodes, based on a set of transmitted or pre-shared issuer certificates. Both possibilities are highlighted in Section 12.1.2.2 and 12.1.2.4.

(2.2) The application then contacts one of the shared computing units, establishes a connection and either matches the target device's certificate (e.g. acknowledged using Transport Layer Security (TLS) with client authentication) with the provided certificate (certificate pinning), or performs a certificate validation to match the issuers with one of the trusted issuer certificates.

(2.3) If the verification and validation completes with a positive result, the target device is trusted, at least for offloading a particular task.

This approach again enables a more centralised approach, relying on the policy decision environment to provide end user certificates, and a more decentralised approach, where the policy decision infrastructure only provides a generic answer. Overall, this workflow is tightly connected with the usage of an external policy decision infrastructure. This is a common environment in corporate use cases, but not available to private users. Chapter 12.3 highlights the policy infrastructure integration into the proposed HMEC framework. This integration also provides ways to hard code simple policies into the application, yielding a simple trust requirement for a specific task. The certificate-based approach enables to manually add trusted nodes for private use cases, e.g. manually adding a user's notebook or home computer, and still make use of this security-enhanced approach without employing policy decision environments.

12.3 Implementation

This chapter focuses on the required client-side extensions to the POWER framework as proposed in Chapter 9 to realise the introduced measures to operate on sensitive data in HMEC use cases. These extensions rely on the existence of a trusted policy evaluation environment, with policies targeted at the specifically executed application on the user's device. Furthermore, a secure enrolment process is required, where (a) the application is bound to one or multiple trusted policy evaluation environments and (b) organisational policies are put into force, to entitle trusted computing unit assessment methods. This chapter provides insights to one possible implementation of the trust mediator, introduced as fundamental part of the architecture in Part III of this thesis.

12.3.1 Enrolment Process

The enrolment process describes necessary steps to be performed before a developed application operating on sensitive data can be used on corporate devices and make use of the HMEC approaches. This involves the definition of policies on different levels, as well as (mutually) binding the application to the policy infrastructures.

Organisational policies need to be defined to apply results of data classifications and risk modelling on the application. This topic is out of the scope of this thesis, but data controllers need to decide, prior enrolment, which of the available device and computing unit attestation methods provide a sufficient level of security. This decision is, for example, influenced by the impact of data loss and the impact of false computation results in the workflow.

The *operational policies* in contrast, describe, on a task level, which operation is allowed to be executed in an offloaded case. This already considers the processed data, the device's environment and the trustworthiness of potential offloading targets. Enrolment of the application on the users' devices without defining operational policies results in completely disabling the HMEC capabilities.

As last step, the policy evaluation environment configuration needs to be embedded in the application. This configuration provides a mapping from a friendly name to the access credentials of the environment. Furthermore, this includes public key material or certificates to verify the integrity and authenticity of issued policy decision responses.

12.3.2 Policy-aware Annotation

The assessment methods and their possible integration in HMEC workflows were already elaborated in Section 12.1.3. The missing link, to form a complete picture, is the interaction of an application developer with the trust mediator and, as a consequence, with the policy evaluation infrastructure.

The first step is the introduction of policy-aware annotations as an extension to the basic annotations. The starting point is illustrated in Listing 12.14.

```
@OffloadMe()
Future method(){
 <code> }
```

Listing 12.14: Basic offloading framework annotation

This instructs the framework and connected components to consider the method as a candidate for offloading as discussed in detail in Part III. The annotation needs to be extended to instruct the offloading framework to (a) invoke the trust mediator component and (b) communicate all potentially necessary data for reference during the policy creation and evaluation process. Therefore, the extended annotation needs to provide the ability to specify the following parameters:

- Specify parameters and context data to be transferred to the policy decision environment.
- Specification of a decision point to use.
- Specification of a unique name, where on one hand the policy creators, and on the other hand the policy cache can refer to.

This results in an extended annotation for the POWER framework as shown in Listing 12.15. Due to its web-based nature and method-based approach, data to be transported can be specified regarding method parameter names, as well as cookie names. The identification of the designated policy decision environment is provided via a *decisionPoint* identifier. This identifier can be resolved to access credentials via an application embedded configuration, added during the enrolment process. In addition, *decisionPointUnreachable* defines a strategy in case the designated decision environment is not reachable. In the example, only *local* execution is allowed, but execution could also be completely denied or only allow execution on certain nodes. Finally, a unique name for identifying this task needs to be provided.

```
@OffloadWithPolicies(
  parametersToSend: ['arg1', 'arg2', 'arg3'],
  cookiesToSend: ['cookie-name1'],
  decisionPoint: 'root',
  decisionPointUnreachable: 'local',
  methodName: 'offloadWithPolicies')
Future offloadWithPolicies(arg1, arg2, {arg3: "test"}){
 <code> }
```

Listing 12.15: Policy-aware annotation

Comparing these extensions to the definition of policies and the defined policy-evaluation-result caching, including the embedded constraints, clearly shows the match between added annotations and how a task's data are transformed to HMEC adapter format and transported to the policy evaluation environment. The following sequence diagram in Figure 12.7 shows a complete cycle starting from application enrolment, over application initialisation, to the actual execution of a certain task, with and without the usage of the policy evaluation cache. To get a complete picture, already discussed interactions like profiling, computing unit discovery, or the concrete device attestations are not included in detail and are kept on an abstract level.

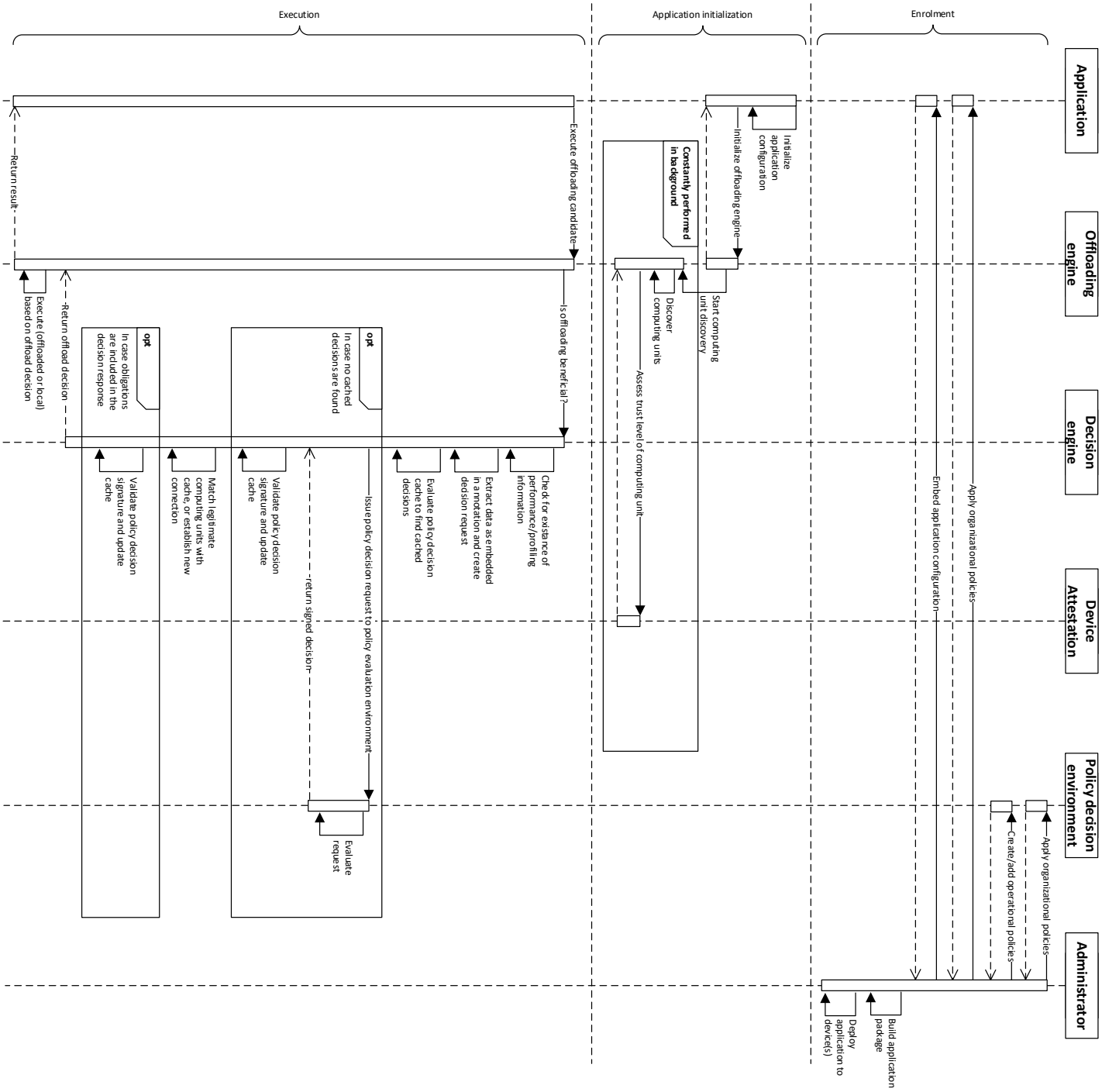


Figure 12.7: Sequence diagram of the offloading process

The sequence diagram is separated into three major parts:

- During the enrolment, the environment and application is prepared. This includes adding organisational and operational policies to the policy decision environment. On the application's side, configuration data is added, and a package is built before being deployed on the user's device.
- In the application initialisation phase, the application is started on the user's device and starts discovering nearby computing units. Depending on the allowed device and computing unit attestation methods, the trustworthiness is determined.
- The execution phase is extended starting from Chapter 9. First a decision request is built according to the defined HMEC format, based on the policy-aware annotation added for the particular task. Next a decision response is acquired, either from the decision response cache, or directly from the policy evaluation environment. After a suitable computing unit was matched to the decision, obligations are enforced and the execution takes place.

The diagram does not go into the very details of each step, as they are already provided in the corresponding section. It rather aims at providing a complete picture of the developed solution and the integration of policy environments especially.

12.4 Performance Evaluation

To evaluate the induced performance overhead of the added policy evaluation capabilities, two policy-related evaluation sets are performed:

- In the first performance-related evaluation the required time to evaluate policies in a standard policy evaluation environment setup is measured.
- In the second evaluation the required time to evaluate cached policies is determined. This evaluation is based on the extended POWER framework as highlighted throughout Chapter 12.3. Still, evaluations are kept on a general level and stay valid also in other environments.

The overall impact on the performance and energy consumption are already discussed in Part III of this thesis and are still valid for policy enabled applications. Performance values determined in this chapter, represent a trade-off between security and performance.

12.4.1 Policy Evaluation Performance at the PDP

The evaluation time of a single request dispatched to the HMEC adapter is a crucial value. High values render HMEC unusable, low values still may have an impact and need to be mitigated by using the discussed advanced caching mechanisms. In this first evaluation, the required time to evaluate policies when requests are dispatched to the policy decision infrastructure is examined. Here, the HMEC framework and mobile device are removed from the scenario to get an unbiased evaluation result.

The test setup comprises of an Amazon EC2 *m4.2xlarge* and *c4.2xlarge* instance running the 64-bit version of Ubuntu 14.04 LTS. The *m4* instance type is a general purpose instance, whereas the *c4* instance type is a data-processing-optimised instance. The use of EC2 instances enables a direct comparison of the execution times with other architectures or other approaches. The evaluation is executed on a modified Apache OpenAZ[3] XACML policy environment, also used as the baseline in several commercial products. The test setup is illustrated in Figure 12.8. The workflow is as follows:

[3] https://github.com/apache/incubator-openaz

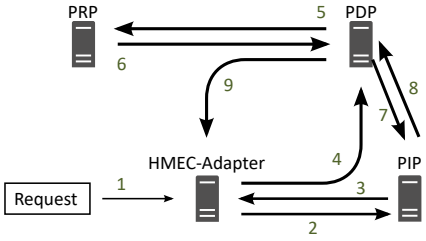


Figure 12.8: Policy evaluation test setup

1. The policy evaluation request is submitted to the HMEC adapter.
2. The HMEC adapter converts the HMEC requests to a XACML conform decision request and contacts the PIP for relevant environment information.
3. The PIP returns corporation dependent environment information like: location of the user, VPN connection state, or on-site presence.
4. The HMEC adapter then enhances the decision request with the received attributes and sends it to the PDP for evaluation.
5. The PDP queries the PRP for matching policies.
6. The PRP determines the matching policies and returns them to the PDP.
7. The PDP may again contact the PIP in case attributes are not resolved yet.
8. The PIP enhances the received request with the required attributes and returns them.
9. The PDP performs the decision process, and returns the decision to the HMEC adapter.
10. The HMEC adapter converts the response to the introduced HMEC format and returns the decision to the client device.

The expected runtime can be estimated by scrutinising on the involved components. The number of policies is considered as the relevant variable in the system, therefore the key factor in influencing the policy evaluation performance. Hence, assessing components with respect to the number of involved policies gives a runtime estimation - n refers to the number of policies:

- *HMEC Adapter:* The HMEC adapter is a converter from one format to another. Its runtime only depends on the incoming request or outgoing response and is not affected by the number policies. Hence, it has a constant time complexity $O(1)$ with respect to the number of policies.
- *PRP:* The PRP needs to return a set of policies eligible on the specified request. Therefore, it needs to walk through all policies and evaluate their applicability. Hence, the PRP has a linear time complexity $O(n)$ with respect to the number of policies.
- *PIP:* The PIP can be seen as a database, where additional attributes can be requested. Without considering a concrete PIP implementation it's hard to provide an accurate estimation. A common implementation for PIPs is based on relational databases. In this case, lookups do not depend on the number of policies but on the number of attributes in total. As a simplification it is assumed that each policy has a constant set of attributes, therefore a direct correlation between number of total attributes and number of policies exist.

  The particular time complexity in relational databases depends on the concrete implementation. As a general guideline it can be assumed that a full table scan (touching each single row) has a

complexity of $O(n)$. Putting appropriate indices on the tables results in a constant to logarithmic time complexity (e.g. organising the index in a B-tree), hence assuming $O(\log n)$.

- *PDP:* The PDP evaluates the policies retrieved from the PRP. The PRP is only capable of doing a pre-selection of the matching policies and does not perform attribute resolution or complex condition evaluation. Therefore, in the worst case, the PRP returns the whole set of available policies. Hence, the PDP has a lower bound time complexity of $O(n)$. Still, for each policy the PDP potentially needs to resolve attributes using the PIP, to verify if a particular policy is matching the request. This results in an overall time complexity of the PDP (and already considering the interactions with PIP) of $O(n \cdot \log n)$.

This results in an expected time complexity as illustrated in Equation 12.1, already neglecting constant time complexity:

$$T(n) = O(n \cdot \log n) \tag{12.1}$$

Following the workflow defined above, the evaluation time for a single policy decision request is recorded for a PRP containing 100, 1000, 10000, and 20000 policies. Real-world use cases deal with tens or hundreds of policies, but also recording corner cases, reveals properties like robustness and scalability of the system. The results of the evaluation are illustrated as a box plot in Figure 12.9 and in tabular representation in in Table 12.1. Results obtained using the *m4* instance are labelled with *m*, results obtained using the *c4* instance are labelled with *c*. At first, it can be observed that the used instance

	100 policies	1000 policies	10000 policies	20000 policies
m	107ms	272ms	1912ms	3995ms
c	88ms	212ms	1663ms	3373ms

Table 12.1: XACML policy evaluation performance results

type only plays a marginal role for the performance. Compared to the expected runtime complexity of $O(n \cdot \log n)$, the evaluation results show a nearly linear time complexity development. This is perfectly in line with the expectations, as the evaluation setup does not require the estimated number of PIP requests. The exemplary policies do not reference unidentified attributes and therefore do not require PIP requests. Excluding PIP requests increases the expressiveness of the evaluation results.

Without policy decision caching enabled this renders applications unusable: Each offloaded task would be delayed by at least additional 88ms.

12.4.2 Policy Caching Performance Evaluation

To evaluate and demonstrate the power of policy caching the average time required to retrieve and match a cached policy is determined. The average evaluation time of a single policy is calculated as an average of evaluating the same policy 100 times. Furthermore, the evaluation considers policies containing data constraints and policies without data constraints.

The evaluation setup comprises of Android and desktop environments. As it was the case in the performance and energy consumption evaluation in Part III, the disparities of different Android devices can be neglected. The devices are running the policy-enabled POWER framework.

The device accessing the cached policies can, in contrast to the general policy evaluation infrastructure, structure the policy cache very efficiently. This results in only a few or single cached policies which need to be checked. As an example: each policy-enabled application part considered for offloading has a unique ID, if policies are structured according to this ID, it is a simple task to rule out most of the cached

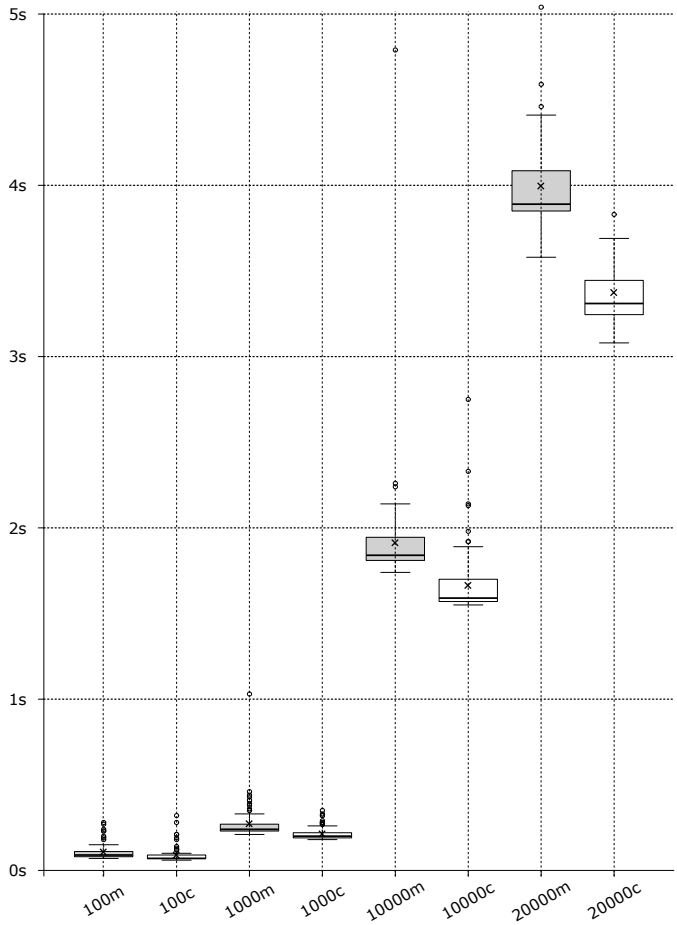


Figure 12.9: XACML policy evaluation performance

policies without even evaluating them in detail. The results are illustrated in Table 12.2. They show the average evaluation time of a single cached policy, with and without embedded constraints.

With policy decision caching and enabling advanced validity constraints, only the first run of an offloaded task is delayed. All subsequent executions are not delayed. Therefore, for policy-enabled applications, caching and enabling validity constraints is a vital feature.

12.5 European General Data Protection Regulation

The European General Data Protection Regulation (GDPR) [European Parliament and the Council of the European Union, 2016] represents a regulation targeting at the protection of EU citizens' personal data as well as the harmonisation of regulations in Europe. It acts as a replacement for the former data protection directive [European Parliament and the Council of the European Union, 1995]. This chapter aims at highlighting the role and relevance of the GDPR for the HMEC processing model and the proposed security mechanisms. Tools and measures provided by the model can be seen as enabling technologies for the implementation of the GDPR but always require an in-depth consideration of the concrete application and the processed data.

The GDPR contains a collection of definitions and obligations for entities including controllers and processors on organisational, but also on technical level. It goes beyond the scope of this thesis to discuss the whole regulation in detail, still the fundamental definitions are important for the following considerations and established links to the HMEC processing model. The GDPR targets the protection

Table 12.2: Average evaluation time of a single cached policy in milliseconds

	Android	Desktop
Test set without constraints	1.71	0.32
Test set with predicates	3.95	1.43

of personal data related to natural persons, stating Article 2 of the regulation:

> *(1) This Regulation applies to the processing of personal data wholly or partly by automated means and to the processing other than by automated means of personal data which form part of a filing system or are intended to form part of a filing system.*
>
> [European Parliament and the Council of the European Union, 2016]

Hence the definition of personal data in Article 4 is an important aspect:

> *(1) 'personal data' means any information relating to an identified or identifiable natural person ('data subject'); an identifiable natural person is one who can be identified, directly or indirectly, in particular by reference to an identifier such as a name, an identification number, location data, an online identifier or to one or more factors specific to the physical, physiological, genetic, mental, economic, cultural or social identity of that natural person;*
>
> [European Parliament and the Council of the European Union, 2016]

Given an application employing the HMEC processing model, and operating on personal data, the application is subject to the GDPR. Thus, also the offloading framework as an enabler for the HMEC processing model is subject to the GDPR. The regulation differentiates between two relevant distinct entities in Article 4: controller and processor

> *(7) 'controller' means the natural or legal person, public authority, agency or other body which, alone or jointly with others, determines the purposes and means of the processing of personal data; where the purposes and means of such processing are determined by Union or Member State law, the controller or the specific criteria for its nomination may be provided for by Union or Member State law;*
>
> *(8) 'processor' means a natural or legal person, public authority, agency or other body which processes personal data on behalf of the controller;*
>
> [European Parliament and the Council of the European Union, 2016]

Controllers hence determine the nature of processed data. In case personal data is processed the GDPR applies. In the context of HMEC, computing units act as processors. The duties of controllers and processors remain unchanged, but HMEC is able to provide enabling technologies. According to Article 24, the controller is responsible to act according to the GDPR and needs to be able to demonstrate that processing is performed in accordance. Therefore, the controller is in charge of selecting appropriate computing units (processors) and establish legal contracts with processors. This process needs to be performed on a per-use case basis and cannot be performed on a general offloading framework level. Hence

the framework cannot relieve the duties off the controller. Still, the framework supports the controller in the implementation phase by providing the attestation capabilities as introduced in Chapter 12.1.3 and highlighted in Chapter 12.6. Using data security policies, controllers can restrict offloading to an agreed set of computing units. It is up to the controller to select appropriate attestation methods. In the simplest case, a certificate-based approach uniquely identifies pre-agreed computing units.

The GDPR is very relevant for the HMEC processing model but is unable to solve organisational obligations or act as a drop-in component for GDPR compliance. The following articles, also containing technical measures and obligations, are most relevant in the scope of the HMEC processing model. These considerations only target at the HMEC framework as part of an application, as the main contribution of this thesis. Measures of the proposed HMEC model and realisation are outlined to support the implementation of the GDPR on a per-use case basis:

- *Article 5 - Principles relating to processing of personal data:* Article 5 settles the scene for processing personal data and defines the basic principles like 'lawfulness', 'fairness and transparency', 'purpose limitation', 'data minimisation', 'accuracy', 'storage limitation' and 'integrity and confidentiality'. Furthermore, it is clarified that the controller is responsible for the compliance with the regulation. Actually, Article 5 is not directly related to the HMEC processing model, but communicates the clear requirement that the controller needs full control of the workflow, data, and utilised computing units. This property is achieved with the proposed model and implementation.

- *Article 17 - Right to erasure ('right to be forgotten'):* The data subject can request the erasure of personal data under the conditions as defined in Article 17.

  Because the HMEC framework potentially processes personal data, which might be requested to be erased, this article is relevant for the HMEC framework. Actually, the HMEC framework does not contain a data storage. Data is passed to the framework and distributed to agreed computing units. No permanent data storage exists, it is only kept in memory for a single offloading operation.

- *Article 18 - Right to restriction of processing:* The proposed implementation might process personal data, subject to processing restrictions. As defined in Article 5 and Article 24, the controller is in charge of ensuring compliance with the regulation and needs to communicate these restrictions, in the form of data security policies, to the offloading framework. This way the HMEC implementation supports the controller in fulfilling the restrictions.

- *Article 25 - Data protection by design and by default:* Article 25 is respected by the HMEC processing model and implementation in several ways.

  - Default settings do not allow offloaded computations. In scenarios where personal data is processed, the controller needs to take action before computational offloading is allowed. The framework adheres to the defined policies for specific tasks. In case no policies are defined, a fallback, non-offloaded execution takes place.
  - Controlled by the data security policies, data transformations can be applied transparently before computational offloading takes place. Encryption, data masking, or anonymisation can be employed to fulfil GDPR requirements on data minimisation and pseudonymisation.

This analysis shows a clear relation of the HMEC processing model to the GDPR. Although the framework cannot act as a drop-in component to ensure compliance, it offers tools to support controllers in adhering to the GDPR. Still, controllers need to evaluate on a per-use case basis the required measures to be applied on the personal data. The proposed HMEC processing model and implementation support the controller by providing adequate technical measures to protect users' personal data.

12.6 Example Scenario: Applications Operating on Personal Data

At this point in this thesis all contributions are discussed and the gained advantages are highlighted. To top the contributions off, this chapter takes exemplary applications, which potentially operate on personal data and shows the possibilities of the framework to adhere to data privacy protection rules, but still offload computations. The selected applications serve the purpose of highlighting the required steps to migrate a standalone application to a computational offloading-enabled application. Each migration process involves one or more of the following actors. Considering the migration of applications operating on personal data as the goal of this chapter, all of them are involved:

- *Controller(s):* An application might process personal data from different controllers. Controllers are responsible to protect the personal data according their data protection directive in force. In context of the application migration, data controllers are responsible to provide data security policies. These policies implement the adherence to the data protection rules.
- *Application Developer:* The application developer is in charge of including the offloading framework into the application and separating the application parts suited for computational offloading from the locally executed application parts. Furthermore, the application developer assigns unique identifiers used during data security policy creation to specifically address data sets.
- *Infrastructure Operator:* The infrastructure operator provides the necessary policy decision infrastructure, which can be referred by the application. For maximum flexibility this infrastructure is provided externally (e.g. as part of a corporation's infrastructure) but can also be embedded in the application.

On a high abstraction level, the following steps are involved in migrating an existing application, in square brackets the involved actor is denoted:

1. *[Controller]* - Determine processed personal data sets.
2. *[Application Developer]* - Include offloading framework and determine parts eligible for offloaded execution.
3. *[Application Developer]* - Enable data security policy evaluation and enforcement for application parts that process personal data sets, as determined beforehand.
4. *[Controller]* - Create data security policies implementing the data protection rules.
5. *[Infrastructure Operator]* - Publish data security policies to policy decision infrastructure.

To illustrate the process, two exemplary applications are selected: an image processing application and a data processing application operating on structured data sets.

12.6.1 Image Processing Application

Assuming an image processing application, which takes pictures e.g. from the local camera, with the ability to apply different modifications or effects on the image. The application developer wants to migrate the application to an offloading-enabled application. In this case the application developer and controller is represented by a single entity. Actually, no personal data is directly involved, but images taken with the user's camera can be categorised as sensitive information. Users may not want their images being processed on public clouds or other untrusted devices. The controller only defines that operations on images require trusted computing units, the decision if a particular computing unit is trustworthy is still up to the user. The application developer/controller categorises data and creates corresponding data security policies, the concrete offloading target is still in hands of the user. The following illustrates the performed steps in more detail following the prior defined notation:

1. *[Controller]* - The controller identifies the processed images as sensitive data which are not allowed to be processed on untrusted computing units.

2. *[Application Developer]* - The application developer includes the offloading framework and optimises identified methods to best suit to the framework. This includes: migrating to asynchronous methods, removing static variable dependencies and optimising the passed arguments (only passing necessary arguments and keeping the offloading operation small in size). This results in methods with a footprint as illustrated in Listing 12.16 or similar.

```
Future applyEffect(effectSelector, image, parameters){
  <code>
}
```

Listing 12.16: Image processing application - method elibigle for offloading

3. *[Application Developer]* - At this stage the application is ready for integration with the offloading framework and its policy decision/enforcement capabilities, hence the introduced annotation is applied. Due to this annotation the policy enforcement is automatically invoked on execution and a suitable computing unit for offloading is selected based on the policy decision. The applied annotation is illustrated in Listing 12.17. The annotation enables to create policies to constrain the selection of computing units.

```
@OffloadWithPolicies(
  parametersToSend: ['effectSelector'],
  cookiesToSend: [],
  decisionPoint: 'root',
  decisionPointUnreachable: 'local',
  methodName: 'applyEffect')
Future applyEffect(effectSelector, image, parameters){
  <code>
}
```

Listing 12.17: Image processing application - policy-aware annotation

4. *[Controller]* - As the next step, the controller can now create concrete data security policies restricting the selection of computing units based on the provided attributes. A controller needs to establish legal contracts with computing units (processors) beforehand to ensure compliance with data protection rules. For this example, as illustrated in Listing 12.18, no parameter-based restrictions are applied. Execution is only allowed on a specific node with the provided identity information. This policy can also be extended to enable execution on any trusted computing node (assessed using the methods described in Chapter 12.1.3), which is then substituted by the client application.

```
<Policy PolicyId="ImageProcessing_applyEffect"
 RuleCombiningAlgId="...:rule-combining-algorithm:deny-overrides">
 <Target/>
 <Rule Effect="Permit" RuleId="AllowOffloadingToTrustedEntities">
  <Target/>
  <ObligationExpressions>
   <ObligationExpression
    ObligationId="urn:power:1.0:obligation:execution-mode"
    FulfillOn="Permit">
    <AttributeAssignmentExpression
     AttributeId="urn:power:1.0:execution-mode">
     <AttributeValue
      DataType="http://www.w3.org/2001/XMLSchema#string">
```

```
        SPECIFIC</AttributeValue>
      </AttributeAssignmentExpression>
      <AttributeAssignmentExpression
       AttributeId="urn:power:1.0:node">
       <Apply
        FunctionId="urn:oasis:names:tc:xacml:1.0:function:string-bag">
        <AttributeValue
         DataType="http://www.w3.org/2001/XMLSchema#string">
         {"endpoint": "client1", "identity": "[CERT-FINGERPRINT]"}
        </AttributeValue>
       </Apply>
      </AttributeAssignmentExpression>
     </ObligationExpression>
    </ObligationExpressions>
   </Rule>
</Policy>
```

Listing 12.18: Data processing application - XACML policy to restrict execution to explicitly trusted nodes

5. *[Infrastructure Operator]* - In the last step the policy is published to the policy infrastructure, which is external to the device or even embedded in the client application.

These measures and extensions enable to operate this image processing application in an offloaded scenario and still protect the user's sensitive data.

12.6.2 Structured Data Processing Application

The second example represents an application operating on structured data sets. It focuses on the personal data aspects and on measures for controllers to protect it. Assuming a simplified application, processing data sets according to the schema defined in Listing 12.19, with the goal to e.g. calculate statistics on the data set.

```
[{ "firstname"      : "...",
   "lastname"       : "...",
   "address"        : "...",
   "city"           : "...",
   "purchase_total" : "..." },...]
```

Listing 12.19: Data processing application - data structure

The data set clearly contains personal data, therefore the data controller is in charge of protecting the data, still the application should be migrated to an offloading-enabled application. Following the same process as defined in Chapter 12.6 different results are achieved because the policies can be created targeting single properties of the processed data set:

1. *[Controller]* - The controller identifies relevant fields in the data structure as personal data. Controllers need to protect personal data according the rules defined in the GDPR.

2. - 3. *[Application Developer]* - Again the application developer includes the offloading framework and applies optimisations as well as annotations as illustrated in Chapter 12.6.1.

4. *[Controller]* - The next step for the controller is to create data security policies that implement the data protection rules. In this step, the controller should minimise the criticality of the transferred data to a minimum and enforce offloaded execution on trusted computing nodes. As a simple example, targeted at the example data scheme, the following can be introduced: Each data entry

represents a single customer's purchase. Given an offloading-enabled method that calculates the overall mean purchase volume, only a single field of the dataset is required. Still, common (and reasonable) practice is, to pass the whole data structure. In offloaded execution scenarios, remote computing units have access to the whole data set, but actually only require access to a small fraction. Listing 12.20 illustrates the automatic eviction of unnecessary field values and minimising the transferred critical data (constraints on allowed execution nodes are omitted here, but are illustrated in Listing 12.18). The relevant path for the eviction in the JSON structure is specified in JSON predicate syntax and gets executed on the client side before offloading to a remote computing unit. Therefore, the remote computing unit only receives the data set with the required information.

```
<Policy PolicyId="DataProcessing_evictPeronalData"
 RuleCombiningAlgId="...:rule-combining-algorithm:deny-overrides">
 <Target/>
 <Rule Effect="Permit" RuleId="AllowOffloadingToTrustedEntities">
  <Target/>
  <ObligationExpressions>
   <ObligationExpression
    ObligationId="urn:power:1.0:client-obligation:json-evict"
    FulfillOn="Permit">
    <AttributeAssignmentExpression AttributeId="urn:power:1.0:client-
        obligation:argument:json-path">
     <Apply
      FunctionId="urn:oasis:names:tc:xacml:1.0:function:string-bag">
      <AttributeValue
       DataType="http://www.w3.org/2001/XMLSchema#string">
       /firstname</AttributeValue>
      <AttributeValue
       DataType="http://www.w3.org/2001/XMLSchema#string">
       /lastname</AttributeValue>
      <AttributeValue
       DataType="http://www.w3.org/2001/XMLSchema#string">
       /address</AttributeValue>
      <AttributeValue
       DataType="http://www.w3.org/2001/XMLSchema#string">
       /city</AttributeValue>
     </Apply>
    </AttributeAssignmentExpression>
   </ObligationExpression>
  </ObligationExpressions>
 </Rule>
</Policy>
```

Listing 12.20: Data processing application - XACML policy to evict unnecessary data

Other strategies are possible like data masking and anonymisation based on regular expressions, tokenisation, also in a format preserving way, or complete removal of unnecessary fields from the data sets.

5. *[Infrastructure Operator]* - In the last step the policy is published to the policy infrastructure, which is external to the device or even embedded in the client application.

The described policy-based approach enables to use existing application logic and represent different use cases. Targeting the data security policy to the particular use case allows to always provide the optimal solution, in terms of protecting users' personal data. The downside of this targeted approach is that a close collaboration between the controller and application developer is required.

12.7 Chapter Conclusions

In this chapter, a clear link was established between the compartmentalised approaches to process sensitive data in (federated) cloud environments and the HMEC approaches. The author of this thesis is the main contributor to the governance model and on the service integration part of the EU SUNFISH project. The SUNFISH governance model obviously serves different domains than HMEC but they have many commonalities. These commonalities are exploited by transforming the federated cloud computing processing model of SUNFISH to the mobile computing world and reusing techniques in the HMEC approach. Furthermore, new challenges are illustrated and solutions are discussed, like the trustworthy assessment of nearby devices and their integration in policy languages. Although no implementation except for the certificate-based approach is provided, clear integration strategies are developed for all attestation techniques. The original SUNFISH approach is developed with XACML in mind to fit into corporation's current environments. For the HMEC adaptation a lightweight and use case targeted alternative for the decision request and response language is proposed to respect the needs of mobile devices but still enable re-usage of existing infrastructures. A different approach could be to define new policy languages and approaches from scratch, which requires users and corporations to provide a separate environment side-by-side with their already existing policy decision and enforcement infrastructure. Still, the defined approaches are only loosely coupled to the concrete policy language and also allow alternative implementations.

By glueing all discussed parts together and providing an integration into the POWER framework, a performance impact analysis is performed. The outcome shows that, although the integration with policy evaluation infrastructures is a step towards centralisation, it is a feasible solution when combined with advanced caching mechanism. The analysis of the GDPR concerning the HMEC processing model and the example scenarios on migrating existing applications clearly show that application developers, administrators, and policy creators need to work hand-in-hand to (a) optimally protect sensitive data, and (b) get the most out of the HMEC approach.

Chapter 13

Conclusions and Future Work

Over the last years, mobile device usage has increased extensively. Mobile devices have become, and are still becoming, more powerful and also enable the execution of computationally intensive applications. Users soon have discovered that mobile devices are limited regarding their available battery power. Making full use of mobile devices' computational potentials has high energy demands and requires to recharge batteries multiple times a day. Industry's solution to this problem is the absolute commitment to cloud computing. Services, deployed in the cloud, are used by mobile devices to store their data and outsource heavy computations using pre-defined interfaces. This approach has major drawbacks regarding flexibility, fail-safety, and data security. Although cloud computing centres have high availability guarantees, the mobile device's Internet connection acts as the weak link. Given a suitable data plan is available, outsourcing heavy computations to the cloud is a suitable solution but has limiting factors, unable to address the needs of future mobile device users.

Researchers in the field of mobile cloud computing converge from a different angle and try to solve the problem using more dynamic approaches. They focus on the application development process and give applications the chance to decide when it is beneficial to use remotely deployed services under current device and environmental conditions. Still, the results only scratch the surface and are tailored to cloud computing and single mobile device platforms. Furthermore, data security and data protection aspects need to be considered in detail to also enable usage for applications operating on sensitive and personal data.

As a first contribution, a threat model is developed based on systematically analysing existing frameworks, resulting in a security requirement catalogue targeted at mobile cloud computing processing models. The evaluation of the security requirements against existing frameworks confirms the first impression that existing frameworks do not consider data security aspects. Therefore, besides contributing a thorough threat model targeting mobile cloud computing, the awareness of data security and data protection gaps in existing computational offloading frameworks is raised. Scrutinising on existing frameworks' architectures reveals a clear focus on client-side functional aspects of offloading frameworks like: optimising the offloading approach or inventing novel partitioning and scheduling algorithms. The server-side is often underspecified and underdeveloped.

The following contributions in this thesis have hence been targeted at removing restrictions from other mobile cloud computing processing models, like the limitation to cloud computing and single mobile device platforms, which results in a novel mobile-first processing model: Hybrid Mobile Edge Computing (HMEC). This model enables to write mobile applications with the ability to dynamically offload computations to other computing units in the mobile device's surrounding or other available computing units (also located in the cloud) and provides a holistic solution. Actually, using the HMEC processing model, the borders between client and server are blurred. All nodes are equal participants of a network, some offering and some consuming computational power. HMEC does not aim at replacing cloud computing but it can be seen as a parallel field of research and development, especially providing

an alternative processing model for mobile applications. Evaluation results show that this processing model has tremendous energy-saving potentials and vast possible performance gains. Although HMEC considers data security and data protection on architectural level, detailed considerations are required.

Data security and data protection is a sensitive topic and requires in-depth analysis, also considering ideas and approaches from different, relevant domains. The author of this thesis was leading the research and development of a data security governance model targeted at federated cloud environments as part of the EU SUNFISH project. The crux of this governance model is the separation of a federation into security compartments, to group services with similar requirements on data security and data protection. Data transfers between the different compartments are controlled by data security policies. The data security policies can completely allow or deny specific operations, but also fine grained decisions are possible. Data transformation services can be employed to selectively encrypt, mask or anonymise particular data fields. In the context of this thesis, clear similarities between the federated cloud and the HMEC processing model are discovered. Hence, it is a logical step to transform the governance model targeted at cloud federations to the mobile world. The integration with the HMEC processing model yields valuable security properties: Although HMEC removes centralised infrastructures, data controllers remain in full control of their data and process flows. Data security policies, in a federated cloud setup as well as in a mobile application HMEC setup can be expressed in XACML syntax, a de-facto standard in the field of data security policy languages. The support for XACML enables operators to re-use their already existing XACML infrastructures and knowledge in a new, mobile-focused environment. The HMEC concept as well as its security foundations, in the context of federated clouds as well as focussed on the transformation to the mobile world, have been exposed to a broad scientific discussion.

A cornerstone and enabler of the HMEC processing model is the deep integration of attestation methods in HMEC's security concept. Modern devices and platforms offer mechanisms to encapsulate executed applications from each other, but also from the owner of the device or platform. The goal of attestation methods is to verify that these shielding and protection mechanisms are still intact, and no one tampered with the device or platform. This way, HMEC is capable of offloading computations, even operating on sensitive data, to unknown devices if attestation succeeds. This gives HMEC an advantage compared to previous processing models and enables usage of many different devices, even in the surrounding of the user. Furthermore, European's General Data Protection Regulation (GDPR), which comes into force in May 2018, entails a "privacy by design" requirement on data controllers, imposing the obligation to implement appropriate technical and organisational measures to protect personal data. HMEC offers the necessary tools to implement these obligations on a per-application basis, on top of data security policies and comply with the regulation, even for computational offloading.

The proposed approaches and solutions are, by far, not the end of the road. As the technological development advances, more sophisticated attestation approaches will be available which need an in-depth consideration. The current implementation relies on the developer to identify parts of the application that should be considered for remote execution. Automatic identification would be preferable but requires more research focussed on the analysis phase. Furthermore, the current solution established interoperability by relying on web technologies, which are not suited for all classes of mobile applications. Maintaining interoperability but still enabling development in a platform's native programming language surely is preferable. Focusing on other developments and related fields of research, the threat model and security requirements catalogue remains valid. After all, research and new developments should be focussed on the users and on protecting their privacy and personal data in accordance with the GDPR.

Part V

Appendices

Appendix A

Peer-to-Peer Networks

This appendix provides a brief introduction to Peer-to-Peer (P2P) networks to the extent as required for understanding the developed models and processes. P2P networks enable to build a network among nodes, where each node can communicate with all other nodes, regardless if they have a direct connection. One characteristic of P2P networks is that they generally do not have a central management point and therefore do not have a central-point-of-failure. The entities in P2P networks can be categorised as *nodes* and *resources*. Nodes represent the participating clients in a network, whereas resources can be considered as services deployed in the network, or data stored in the network. Various different flavors of P2P networks exist. Reiter [2015] provides a categorisation of P2P networks into unstructured and structured networks.

Unstructured networks are considered as the first generation of P2P frameworks. Using this type of network, nodes randomly connect to a pre-defined number of existing peers without taking into account the structure of the network. Other peers are discovered by flooding the network with discovery requests, and further connections are built to random nodes in the network. The unstructured approach also continues on the resource level. Each node can offer resources, but other nodes need to flood the network with search requests to discover the resources. This results in a massive amount of broadcasted search requests throughout the whole network. Mapping the topology of an unstructured P2P network to a tree could result in a highly unbalanced tree negatively impacting performance in the network. An unbalanced tree affects the connectivity in the network and may result in a situation where peers may not be aware of the shared resources of a distant node. Anyhow, unstructured P2P networks were successfully used for a long time for file-sharing. The Gnutella[1] network, for example, uses an unstructured approach.

Structured P2P networks are the next improved generation. In contrast to unstructured networks, structured networks follow a well-defined algorithm to maintain the topology of the network. Furthermore, resources are deeply integrated into the network and each node at least is aware of the resources provided by its neighbor nodes. One implementation concept to realise structured P2P networks is based on DHTs. Basically, DHTs provide a key-value mapping like local hash tables but distributed over the whole network.

DHTs make a distinction between node IDs and key IDs but are in the same address space. Both are unique but serve a different purpose. The key IDs act, like in a local hash table scenario, as an identifier to retrieve a previously saved value or to save a new value. Node IDs uniquely identify a single node where actual values are stored. The DHT implementation provides a mapping mechanism to determine on which nodes a particular key-value pair is stored. Chord [Stoica et al., 2001] uses this approach to realise a scalable P2P lookup service. In the Chord approach, nodes are distributed in the address space by hashing a node's IP address and are arranged in a circle. Key IDs are placed in the same address space. Each node is responsible for all key IDs greater than the node ID of its predecessor but smaller or

[1] https://courses.cs.washington.edu/courses/cse522/05au/gnutella_protocol_0.4.pdf

equal to the local node ID. This way a key distribution over all nodes is achieved. Other, slightly different approaches like Pastry [Rowstron and Druschel, 2001; Lua et al., 2005] or CAN [Lua et al., 2005] differ in the details of assigning node IDs and in finding routes to nodes in charge of a specified key ID.

Figure A.1 illustrates the basic concept of DHTs. The numbered circles can equally be key IDs or node IDs, whereas the number illustrates the unique ID and the arrows between the circles illustrate established connections. The DHT is partitioned into several areas, illustrated as shaded areas, where the information of multiple IDs is cumulated. Each node in a certain area saves the whole information of its area to increase availability and redundancy. The connections are used for routing requests. Due to the structure of the P2P network, a request is sent to the connection with the closest ID. To foster simplicity, Figure A.1 shows a very basic example, where each node only has two neighbor connections, real-world implementations will add more connections to distant nodes to decrease hop count. A typical routing algorithm in this type of network can be sketched as follows:

1. Extract target ID of the incoming request.
2. Check if the local node or one of the local resource IDs matches incoming target ID. If nothing is found continue with next step.
3. Iterate through all connected nodes and find the one with the closest ID.
4. If the distance from target ID to closest ID is larger than the distance from local ID to target ID cancel processing, the ID was not found.
5. Send request to closest node.

In contrast to unstructured networks, a hop counter is not required in structured P2P networks, because the distance to the target ID is decreased in every step and therefore already visited nodes are not considered multiple times. The routing performance is increased with distributing the established connections to distant and near nodes in the network.

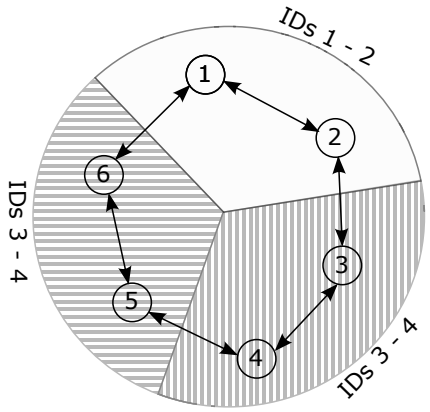


Figure A.1: Concept of structured P2P networks based on distributed hash tables [Reiter, 2015]

Bibliography

Abolfazli, Saeid, Abdullah Gani, and Min Chen [2015]. "HMCC: A Hybrid Mobile Cloud Computing Framework Exploiting Heterogeneous Resources". In: *2015 3rd IEEE International Conference on Mobile Cloud Computing, Services, and Engineering*. 2015, pages 157–162. ISBN 978-1-4799-8977-5. doi:10.1109/MobileCloud.2015.28. http://ieeexplore.ieee.org/xpl/articleDetails.jsp?arnumber=7130881 (cited on page 20).

Agarwal, Sharad and Jacob R. Lorch [2009]. "Matchmaking for online games and other latency-sensitive P2P systems". *ACM SIGCOMM Computer Communication Review* 39.4 (2009), page 315. ISSN 01464833. doi:10.1145/1594977.1592605 (cited on page 65).

Anati, Ittai, Shay Gueron, Simon Johnson, and Vincent Scarlata [2013]. "Innovative Technology for CPU Based Attestation and Sealing". *HASP - Proceedings of the 2nd International Workshop on Hardware and Architectural Support for Security and Privacy* (2013), pages 1–7. doi:10.1.1.405.8266 (cited on page 117).

Bartel, Mark, John Boyer, Barb Fox, Brian LaMacchia, and Ed Simon [2008]. *XML Signature Syntax and Processing (Second Edition)*. 2008. https://www.w3.org/TR/xmldsig-core/ (cited on page 112).

Baset, Salman A. and Henning G. Schulzrinne [2006]. "An analysis of the Skype peer-to-peer internet telephony protocol". In: *Proceedings of IEEE INFOCOM*. 2006. ISBN 1424402212. doi:10.1109/INFOCOM.2006.312. arXiv: 0412017 [cs] (cited on page 80).

Baumann, Andrew, Marcus Peinado, and Galen Hunt [2014]. "Shielding Applications from an Untrusted Cloud with Haven". *Proceedings of the 11th USENIX Conference on Operating Systems Design and Implementation* (2014), pages 267–283. ISSN 0734-2071. doi:10.1145/2799647. http://dl.acm.org/citation.cfm?id=2685048.2685070 (cited on page 119).

Beckman, Peter H [2005]. "Building the TeraGrid." *Philosophical transactions. Series A, Mathematical, physical, and engineering sciences* 363.1833 (2005), pages 1715–28. ISSN 1364-503X. doi:10.1098/rsta.2005.1602. http://www.ncbi.nlm.nih.gov/pubmed/16099743 (cited on page 10).

Begtašević, F and P Van Mieghem [2000]. "Measurements of the Hopcount in Internet" (2000), pages 1–12 (cited on page 64).

Bernholdt, David, Shishir Bharathi, David Brown, Kasidit Chanchio, Meili Chen, Ann Chervenak, Luca Cinquini, Bob Drach, Ian Foster, Peter Fox, Jose Garcia, Carl Kesselman, Rob Markel, Don Middleton, Veronika Nefedova, Line Pouchard, Arie Shoshani, Alex Sim, Gary Strand, and Dean Williams [2005]. "The Earth System Grid: Supporting the next generation of climate modeling research". *Proceedings of the IEEE* 93.3 (2005), pages 485–495. ISSN 00189219. doi:10.1109/JPROC.2004.842745. arXiv: 0712.2262 (cited on page 10).

Bray, Tim, Dave Hollander, Andrew Layman, Richard Tobin, and Henry S. Thompson [2009]. *Namespaces in XML 1.0 (Third Edition)*. 2009. http://www.w3.org/TR/2009/REC-xml-names-20091208/ (cited on page 110).

Burgstaller, Florian, Andreas Derler, Stefan Kern, Gabriel Schanner, and Andreas Reiter [2016]. "Anonymous Communication in the Browser via Onion-Routing". *Proceedings - 2015 10th International Conference on P2P, Parallel, Grid, Cloud and Internet Computing, 3PGCIC 2015* (2016), pages 260–267. doi:10.1109/3PGCIC.2015.22 (cited on page 79).

Catalano, Dario and Dario Fiore [2013]. "Practical Homomorphic MACs for Arithmetic Circuits". In: *EUROCRYPT 2013*. 2013, pages 336–352 (cited on page 21).

Chun, Byung-Gon, Sunghwan Ihm, Petros Maniatis, Mayur Naik, and Ashwin Patti [2011]. "Clonecloud: Elastic Execution Between Mobile Device and Cloud". In: *Proceedings of the sixth conference on Computer systems*. 2011, pages 301–314. ISBN 9781450306348. http://dl.acm.org/ft%7B%5C_%7Dgateway.cfm?id=1966473%7B%5C&%7Dtype=pdf (cited on pages 27, 29, 30, 50, 51, 53).

Costan, Victor and Srinivas Devadas [2016]. "Intel SGX Explained". *Cryptology ePrint Archive, Report 2016/086* (2016) (cited on page 117).

Coughlin, Tom [2015]. "A Moore's Law for Mobile Energy". *IEEE Consumer Electronics Magazine* 4.1 (2015), pages 74–82 (cited on page 4).

Cuervo, Eduardo, Aruna Balasubramanian, Dae-ki Cho, Alec Wolman, Saroiu Stefan, Ranveer Chandra, and Bahl Paramvir [2010]. "MAUI : Making Smartphones Last Longer with Code Offload". In: *Proceedings of the 8th international conference on Mobile systems, applications, and services*. Volume 17. 2010, pages 49–62. ISBN 9781605589855. http://dl.acm.org/citation.cfm?id=1814441 (cited on pages 27–29, 50, 51, 54, 64, 74, 75, 87, 89).

Dias, David [2015]. "browserCloud.js - A federated community cloud served by a P2P overlay network on top of the web platform". Master Thesis. INESC-ID Lisbon, 2015. https://github.com/diasdavid/thesis.browserCloud.js/blob/master/document.pdf (cited on page 79).

Enck, William, Machigar Ongtang, and Patrick McDaniel [2009]. "Understanding Android Security". *IEEE Security and Privacy* 7 (2009), pages 50–57. ISSN 1540-7993. doi:10.1109/MSP.2009.26. arXiv: 1512.00567 (cited on page 61).

ETSI [2015]. "ETSI GS MEC-IEG 004 - Mobile Edge Computing (MEC); Service Scenarios" (2015) (cited on page 15).

ETSI [2016a]. "ETSI GS MEC 002 - Mobile Edge Computing (MEC); Technical Requirements" (2016). http://www.etsi.org/standards-search (cited on page 14).

ETSI [2016b]. "ETSI GS MEC 003 - Mobile Edge Computing (MEC); Framework and Reference Architecture" (2016) (cited on page 14).

ETSI [2017]. "ETSI GS MEC-IEG 006 - Mobile Edge Computing; Market Acceleration; MEC Metrics Best Practice and Guidelines" (2017) (cited on page 14).

European Parliament and the Council of the European Union [1995]. "Directive 95/46/EC of the European Parliament and of the Council of 24 October 1995 on the protection of individuals with regard to the processing of personal data and on the free movement of such data". *Official Journal of the European Union* L 281 (1995). http://eur-lex.europa.eu/LexUriServ/LexUriServ.do?uri=CELEX:31995L0046:EN:HTML (cited on page 127).

European Parliament and the Council of the European Union [2016]. "Regulation (EU) 2016/679 of the European Parliament 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* L119 (2016) (cited on pages 127, 128).

Faruki, Parvez, Ammar Bharmal, Vijay Laxmi, Vijay Ganmoor, Manoj Singh Gaur, Mauro Conti, and Muttukrishnan Rajarajan [2015]. "Android Security: A Survey of Issues, Malware Penetration, and Defenses". *IEEE Communications Surveys & Tutorials* 17.2 (2015), pages 998–1022. ISSN 1553-877X. doi:10.1109/COMST.2014.2386139. http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=6999911 (cited on page 61).

Fenstermacher, David, Craig Street, Tara McSherry, Vishal Nayak, Casey Overby, and Michael Feldman [2005]. "The Cancer Biomedical Informatics Grid (caBIGTM)." *Conference proceedings of the IEEE Engineering in Medicine and Biology Society* 27 (2005), pages 743–746. ISSN 1557-170X. doi:10.1109/IEMBS.2005.1616521 (cited on page 10).

Fernando, Niroshinie, Seng W. Loke, and Wenny Rahayu [2012]. "Mobile cloud computing: A survey". *Future Generation Computer Systems* 29.1 (2012), pages 84–106. ISSN 0167739X. doi:10.1016/j.future.2012.05.023. http://dx.doi.org/10.1016/j.future.2012.05.023 (cited on pages 19, 49).

Fette, I. and A. Melnikov [2011]. *The WebSocket Protocol*. Technical report. RFC 6455, RFC Editor, 2011. https://www.rfc-editor.org/info/rfc6455 (cited on pages 69, 78, 80).

Foster, Ian, Yong Zhao, Ioan Raicu, and Shiyong Lu [2008]. "Cloud computing and grid computing 360-degree compared". *Grid Computing Environments Workshop 2008 (GCE '08)* (2008), pages 1–10. doi:10.1109/GCE.2008.4738445. arXiv: 0901.0131. http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=4738445 (cited on page 11).

Gennaro, Rosario, Craig Gentry, and Bryan Parno [2010]. "Non-interactive verifiable computing: Outsourcing computation to untrusted workers". *Advances in Cryptology–CRYPTO* (2010), pages 465–482. ISSN 03029743. doi:10.1007/978-3-642-14623-7_25 (cited on page 21).

Gennaro, Rosario, Craig Gentry, Bryan Parno, and Mariana Raykova [2013]. "Quadratic Span Programs and Succinct NIZKs without PCPs". 1017660 (2013), pages 626–645 (cited on page 21).

Google [2017]. *Android Platform Versions; Accessed on 19.2.* 2017. https://developer.android.com/about/dashboards/index.html (cited on pages 30, 31).

Gordon, Mark S, David Ke Hong, Peter M Chen, Jason Flinn, Scott Mahlke, and Zhuoqing Morley Mao [2015]. "TANGO: Accelerating Mobile Applications through Flip-Flop Replication". *GetMobile* 19.3 (2015), pages 10–13 (cited on pages 27, 33, 51).

Goyal, Sachin and John Carter [2004]. "A Lightweight Secure Cyber Foraging Infrastructure for Resource-Constrained Devices". In: *Proceedings of the Sixth IEEE Workshop on Mobile Computing Systems and Applications (WMCSA 2004)*. 6. 2004 (cited on pages 19, 53).

Ha, Kiryong, Padmanabhan Pillai, Wolfgang Richter, Yoshihisa Abe, and Mahadev Satyanarayanan [2013]. "Just-in-Time Provisioning for Cyber Foraging" (2013), pages 153–166 (cited on page 53).

Höner, Patrick [2013]. "Cloud Computing Security Requirements and Solutions : a Systematic Literature Review" (2013) (cited on page 40).

Horsmanheimo, Seppo, Niwas Maskey, and Lotta Tuomimäki [2014]. "Feasibility study of utilizing mobile communications for smart grid applications in urban area" (2014), pages 446–451 (cited on page 54).

Hu, Yun Chao, Milan Patel, Dario Sabella, Nurit Sprecher, and Valerie Young [2015]. "Mobile Edge Computing - A key technology towards 5G". *ETSI White Paper* 11 (2015). http://www.etsi.org/images/files/ETSIWhitePapers/etsi%7B%5C_%7Dwp11%7B%5C_%7Dmec%7B%5C_%7Da%7B%5C_%7Dkey%7B%5C_%7Dtechnology%7B%5C_%7Dtowards%7B%5C_%7D5g.pdf (cited on page 14).

Iankoulova, Iliana [2011]. "Cloud Computing Security Requirements : a Systematic Review" (2011) (cited on page 40).

Jacob, Bart, Michael Brown, Kentaro Fukui, and Nihar Trivedi [2005]. *Introduction to Grid Computing*. First Edit. IBM, 2005, page 262. ISBN 9780738494005. http://www.redbooks.ibm.com/abstracts/sg246778.html?Open (cited on pages 9, 10).

Jones, Michael, John Bradler, and Nat Sakimura [2015]. *JSON Web Token (JWT)*. Technical report. RFC Editor, 2015. doi:2070-1721. http://www.rfc-editor.org/rfc/rfc7519.txt (cited on page 112).

Kemp, Roelof, Nicholas Palmer, Thilo Kielmann, and Henri Bal [2012]. "Cuckoo: a computation offloading framework for smartphones". In: *Mobile Computing, Applications, and Services. Second International ICST Conference, MobiCASE 2010*. 2012, pages 59–79. http://www.springerlink.com/index/U6777405301NP063.pdf (cited on pages 27, 30, 32, 53).

Kosta, Sokol, Andrius Aucinas, Pan Hui, Richard Mortier, and Xinwen Zhang [2012]. "ThinkAir: Dynamic resource allocation and parallel execution in the cloud for mobile code offloading". In: *2012 Proceedings IEEE INFOCOM*. Ieee, Mar. 2012, pages 945–953. ISBN 978-1-4673-0775-8. doi:10.1109/INFCOM.2012.6195845. http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=6195845 (cited on pages 27, 32, 33, 53, 89).

Laure, Erwin, F. Hemmer, A. Aimar, M. Barroso, P. Buncic, A. Di Meglio, L. Guy, P. Kunszt, S. Beco, F. Pacini, F. Prelz, M. Sgaravatto, A Edlund, O. Mulmo, D. Groep, S.M. Fisher, and M. Livny [2004]. "Middleware for the next generation Grid infrastructure". *Proceedings of the International Conference on Computing in High Energy and Nuclear Physics (CHEP'04)* (2004), pages 826–830. doi:10.5170/CERN-2005-002.826 (cited on page 11).

Lee, Jae-nam, Minh Q Huynh, Ron Chi-Wai Kwok, and Shih-ming Pi [2003]. "IT outsourcing evolution: past, present, and future". *Communications of the ACM* 46.5 (2003), pages 84–89. ISSN 00010782. doi:10.1145/769800.769807 (cited on page 3).

Lee, Youngki, Sharad Agarwal, Chris Butcher, and Jitu Padhye [2008]. "Measurement and estimation of network QoS among peer Xbox 360 game players". *Lecture Notes in Computer Science* 4979 LNCS (2008), pages 41–50. ISSN 03029743. doi:10.1007/978-3-540-79232-1{_}5 (cited on pages 65, 79).

Lewis, Grace A., Sebastian Echeverría, Soumya Simanta, Ben Bradshaw, and James Root [2014]. "Cloudlet-Based Cyber-Foraging for Mobile Systems in Resource-Constrained Edge Environments". In: *Proceedings of the 36th International Conference on Software Engineering*. 2014, pages 412–415. ISBN 9781450327688 (cited on page 19).

Lin, Boci, Yan Chen, Xu Chen, and Yingying Yu [2012]. "Comparison between JSON and XML in Applications Based on AJAX". *Proceedings - 2012 International Conference on Computer Science and Service System* (2012), pages 1174–1177. doi:10.1109/CSSS.2012.297 (cited on page 109).

Lua, Eng Keong, Jon Crowcroft, Pias Marcelo, Ravi Sharma, and Lim Steven [2005]. "A Survey and Comparison of Peer-to-Peer Overlay Network Schemes". *IEEE Communications Surveys & Tutorials* (2005), pages 72–93. ISSN 14388871. doi:10.2196/jmir.1752 (cited on page 140).

Luan, Tom H., Longxiang Gao, Zhi Li, Yang Xiang, Guiyi We, and Limin Sun [2016]. "Fog Computing: Focusing on Mobile Users at the Edge". *eprint arXiv:1502.01815v3* (2016), pages 1–11. ISSN 10848045. doi:10.1016/j.jnca.2015.02.002. arXiv: 1502.01815. http://arxiv.org/abs/1502.01815 (cited on pages 13, 14).

Lui, F, J. Tong, J. Mao, R. Bohn, J. Messina, L. Badget, and D. Leaf [2011]. "NIST Cloud Computing Reference Architecture: Recommendations of the National Institute of Standards and Technology". *NIST Special Publication 500-292* (2011). doi:500-299 (cited on pages 11–13).

Mahy, Y, P Matthews, and J Rosenberg [2010]. *Traversal Using Relays around NAT (TURN): Relay Extensions to Session Traversal Utilities for NAT (STUN)*. 2010. https://tools.ietf.org/html/rfc5766 (cited on page 65).

NIST [2011]. “Cloud Architecture Reference Models: A Survey”. *Document NIST CCRATWG 004* (2011) (cited on page 11).

Nurseitov, Nurzhan, Michael Paulson, Randall Reynolds, and Clemente Izurieta [2009]. “Comparison of JSON and XML Data Interchange Formats: A Case Study”. *Caine* (2009), pages 157–162. http://www.cs.montana.edu/izurieta/pubs/caine2009.pdf (cited on page 109).

OASIS [2013]. *eXtensible Access Control Markup Language (XACML) Version 3.0*. 2013. http://docs.oasis-open.org/xacml/3.0/xacml-3.0-core-spec-os-en.html (cited on pages 95, 99).

OASIS [2014]. *JSON Profile of XACML 3.0 Version 1.0*. 2014. http://docs.oasis-open.org/xacml/xacml-json-http/v1.0/xacml-json-http-v1.0.html (cited on pages ix, 109, 110).

Parno, Bryan, Jon Howell, Craig Gentry, and Mariana Raykova [2013]. “Pinocchio: Nearly practical verifiable computation”. *Proceedings - IEEE Symposium on Security and Privacy* (2013), pages 238–252. ISSN 10816011. doi:10.1109/SP.2013.47 (cited on page 21).

Pordes, Ruth, Don Petravick, Bill Kramer, Doug Olson, Miron Livny, Alain Roy, Paul Avery, Kent Blackburn, Torre Wenaus, Frank Würthwein, Ian Foster, Rob Gardner, Mike Wilde, Alan Blatecky, John McGee, and Rob Quick [2007]. “The open science grid”. *Journal of Physics: Conference Series* 78 (2007), page 012057. ISSN 1742-6588. doi:10.1088/1742-6596/78/1/012057 (cited on page 10).

Reiter, Andreas [2015]. “Enabling Secure Communication over Existing Peer-to-Peer Frameworks”. *2015 23rd Euromicro International Conference on Parallel, Distributed, and Network-Based Processing* (2015), pages 575–582. ISSN 1066-6192. doi:10.1109/PDP.2015.10. http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=7092777 (cited on pages 6, 139, 140).

Reiter, Andreas [2017]. “Secure Policy-based Device-to-Device Offloading for Mobile Applications”. In: *32nd ACM Symposium on Applied Computing*. In Press, 2017. ISBN 9781450344869 (cited on pages 6, 103).

Reiter, Andreas and Alexander Marsalek [2017]. “WebRTC: Your Privacy is at Risk”. *32nd ACM Symposium on Applied Computing* In press (2017) (cited on pages 6, 7).

Reiter, Andreas, Bernd Prünster, and Thomas Zefferer [2017]. “Hybrid Mobile Edge Computing: Unleashing the Full Potential of Edge Computing in Mobile Device Use Cases”. In: *2017 17th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGrid)*. In Press, 2017 (cited on pages 6, 7, 20).

Reiter, Andreas and Thomas Zefferer [2015a]. “Paving the Way for Security in Cloud-Based Mobile Augmentation Systems”. In: *3rd IEEE International Conference on Mobile Cloud Computing, Services, and Engineering (IEEE Mobile Cloud 2015)*. 2015, pages 89–98. ISBN 978-1-4799-8977-5. doi:10.1109/MobileCloud.2015.12. http://ieeexplore.ieee.org/xpl/articleDetails.jsp?arnumber=7130873 (cited on pages 6, 27, 51).

Reiter, Andreas and Thomas Zefferer [2015b]. “POWER: A cloud-based mobile augmentation approach for web- and cross-platform applications”. *2015 IEEE 4th International Conference on Cloud Networking, CloudNet 2015* (2015), pages 226–231. doi:10.1109/CloudNet.2015.7335313 (cited on pages 6, 73).

Reiter, Andreas and Thomas Zefferer [2016]. “Flexible and Secure Resource Sharing for Mobile Augmentation Systems”. *2016 4th IEEE International Conference on Mobile Cloud Computing, Services, and Engineering (MobileCloud)* (2016), pages 31–40. doi:10.1109/MobileCloud.2016.8. http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=7474403 (cited on pages 6, 52, 56, 78, 84, 86–90).

Rhee, Keunwoo, Dongho Won, Sang Woon Jang, Sooyoung Chae, and Sangwoo Park [2013]. "Threat modeling of a mobile device management system for secure smart work". *Electronic Commerce Research* 13.3 (2013), pages 243–256. ISSN 13895753. doi:10.1007/s10660-013-9121-4 (cited on pages 113, 114).

Robie, Jonathan, Michael Dyck, and Josh Spiegel [2017]. *XML Path Language (XPath)*. 2017. https://www.w3.org/TR/2017/PR-xpath-31-20170117/ (cited on page 110).

Rong, Chunming, Son T. Nguyen, and Martin Gilje Jaatun [2012]. "Beyond lightning: A survey on security challenges in cloud computing". *Computers & Electrical Engineering* 39.1 (Jan. 2012), pages 47–54. ISSN 00457906. doi:10.1016/j.compeleceng.2012.04.015. http://linkinghub.elsevier.com/retrieve/pii/S0045790612000870 (cited on page 37).

Rosenberg, J [2010]. *Interactive Connectivity Establishment (ICE): A Protocol for Network Address Translator (NAT) Traversal for Offer/Answer Protocols*. Technical report. 2010. https://tools.ietf.org/html/rfc5245 (cited on page 65).

Rowstron, Antony and Peter Druschel [2001]. "Pastry: Scalable, Decentralized Object Location, and Routing for Large-Scale Peer-to-Peer Systems". *Middleware 2001* 2218.November 2001 (2001), pages 329–350. ISSN 03029743. doi:10.1007/3-540-45518-3. http://www.springerlink.com/index/10.1007/3-540-45518-3 (cited on page 140).

Ryan, Mark D. [2013]. "Cloud computing security: The scientific challenge, and a survey of solutions". *Journal of Systems and Software* 86.9 (Sept. 2013), pages 2263–2268. ISSN 01641212. doi:10.1016/j.jss.2012.12.025. http://www.sciencedirect.com/science/article/pii/S0164121212003378 (cited on page 37).

Saiedian, Hossein and Dan Broyle [2011]. "Security vulnerabilities in the same-origin policy: Implications and alternatives". *IEEE Computer Society* 44.9 (2011), pages 29–36. ISSN 00189162. doi:10.1109/MC.2011.226 (cited on page 78).

Satyanarayanan, Mahadev [2001]. "Pervasice Computing: Vision and Challenges". *IEEE Personal Communications* (2001), pages 10–17 (cited on pages 17–19).

Schmuck, F and R Haskin [2002]. "GPFS: A Shared-Disk File System for Large Computing Clusters". *Proceedings of the First USENIX Conference on File and Storage Technologies* January (2002), pages 231–244. http://portal.acm.org/citation.cfm?id=1083349 (cited on page 10).

Shepler, Spencer, Mike Eisler, David Robinson, Brent Callaghan, Robert Thurlow, David Noveck, and Carl Beame [2003]. *Network File System (NFS) Version 4 Protocol*. Technical report. RFC Editor, 2003. http://www.rfc-editor.orf/rfc/rfc3530.txt (cited on page 10).

Sinha, Kanad and Milind Kulkarni [2011]. "Techniques for Fine-Grained, Multi-site Computation Offloading". *2011 11th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing* (May 2011), pages 184–194. doi:10.1109/CCGrid.2011.69. http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=5948609 (cited on page 51).

Stoica, Ion, Robert Morris, David Karger, M. Frans Kaashoek, and Hari Balakrishnan [2001]. "Chord: A Scalable Peer-to-Peer Lookup Service for Internet Applications". *Conference on Applications, technologies, architectures, and protocols for computer communications (SIGCOMM '01)* (2001), pages 149–160. ISSN 01464833. doi:10.1145/383059.383071. http://portal.acm.org/citation.cfm?doid=383059.383071 (cited on pages 64, 139).

Sumaray, Audie and S. Kami Makki [2012]. "A Comparison of Data Serialization Formats for Optimal Efficiency on a Mobile Platform". *Proceedings of the 6th International Conference on Ubiquitous Information Management and Communication* 8.4 (2012), page 1. ISSN 1559-1131. doi:10.1145/1281700.1281707. http://dl.acm.org/citation.cfm?id=2184810 (cited on page 109).

Suzic, Bojan and Andreas Reiter [2016]. “Towards Secure Collaboration in Federated Cloud Environments”. *Workshop on Security, Privacy, and Identity Management in the Cloud* (2016) (cited on pages 6, 100).

TomP2P [2014]. *TomP2P: A P2P-based high performance key-value pair storage library*. 2014. http://tomp2p.net (cited on page 80).

Yan, Zheng, Xixun Yu, and Wenxiu Ding [2017]. “Context-aware verifiable cloud computing”. *IEEE Access* 5 (2017), pages 2211–2227. ISSN 21693536. doi:10.1109/ACCESS.2017.2666839 (cited on page 21).

Yu, Xixun, Zheng Yan, and Athanasios V. Vasilakos [2017]. “A Survey of Verifiable Computation”. *Mobile Networks and Applications* 22.3 (2017), pages 438–453. ISSN 15728153. doi:10.1007/s11036-017-0872-3 (cited on page 21).

Zhang, Lide, Birjodh Tiwana, Zhiyun Qian, Zhaoguang Wang, Robert P Dick, Z Morley Mao, and Lei Yang [2010]. “Accurate Online Power Estimation and Automatic Battery Behavior Based Power Model Generation for Smartphones”. *Proceedings of the eighth IEEE/ACM/IFIP international conference on Hardware/software codesign and system synthesis* (2010), pages 105–114 (cited on page 86).

Zhou, Bowen, Amir Vahid Dastjerdi, Rodrigo N Calheiros, Satish Narayana Srirama, and Rajkumar Buyya [2015]. “mCloud: A Context-aware Offloading Framework for Heterogeneous Mobile Cloud”. 1374.c (2015), pages 1–14. doi:10.1109/TSC.2015.2511002 (cited on pages 27, 34, 54).