Secure Policy-based Device-to-Device Offloading for Mobile Applications

Andreas Reiter
Institute of Applied Information Processing and Communications (IAIK)
Graz University of Technology
Inffeldgasse 16a, 8010 Graz, Austria
andreas.reiter@iaik.tugraz.at

ABSTRACT

Mobile application offloading, with the purpose of extending battery lifetime and increasing performance has been intensively discussed recently, resulting in various different solutions: mobile device clones operated as virtual machines in the cloud, simultaneously running applications on the mobile device and on a distant server, as well as flexible solutions dynamically acquiring other mobile devices' resources in the user's surrounding. Existing solutions have gaps in the fields of data security and application security. These gaps can be closed by integrating data usage policies, as well as application-flow policies. In this paper, we propose and evaluate a novel approach of integrating XACML into existing mobile application offloading-frameworks. Data owners remain in full control of their data, still, technologies like device-to-device offloading can be used.

CCS Concepts

•Information systems → Mobile information processing systems; Web applications; •Security and privacy → Web application security; •Human-centered computing → Ubiquitous and mobile computing; •Social and professional topics → Privacy policies;

Keywords

Mobile Application Offloading; Mobile Cloud Computing; Hybrid Mobile Cloud Computing; XACML; Data Security; Governance; Mobile Device Attestation

1. MOTIVATION, VISION, AND GOALS

Mobile devices are excessively used to complete all kinds of tasks throughout the day, regardless if tasks are work related or private tasks. The usage of mobile devices has already overtaken the usage of desktop computers in 2011[1]. Since then, more and more applications like office apps or image processing apps are ported to mobile devices as native applications or using cross-platform frameworks. Recently, it was not possible to execute these computationally intensive applications on mobile devices due to very limited computing resources. The ever increasing power, larger displays, and enhanced sensors also enable the execution of these applications on mobile devices. Reasons for this trend are clear: mobile devices enable access to information independent of users' location, at any time. Still, a blocking factor on high-end mobile devices is battery life. Heavily utilizing the mobile device's CPU for a longer period of time, drains a lot of battery and requires the user to recharge its phone multiple times a day. To counter this issue, technologies like Surrogate Computing [**?**] or Mobile Cloud Computing (MCC) [**?**] were introduced. These technologies use computational resources from other devices or clouds to save energy and improve performance.

All existing frameworks targeting these technologies achieve good results in their specific environment. Nevertheless, all of them contribute on how to perform the actual offloading operation and how to identify application-parts suitable for offloading. Definitely, these are important aspects of an application offloading framework, but other important factors were not considered so far: maintaining data security and control of the data owner over its data and involved processes. Especially in corporate use cases, strict requirements on data security and data-flow adhere. Assurance is required that sensitive data is not transferred to untrusted devices, or outside of specific regions. Furthermore, complete and explicit control of the data-flows is required.

Contributions. The contributions of this work are twofold. First, a flexible and security-focused architecture for mobile application offloading frameworks is introduced. The architecture closes the gaps identified during examination of existing frameworks. Second, the implementation of an existing framework is extended and new components are introduced, to realize a mobile-application offloading-framework which gives data- and process owners full control.

Acknowledgments. This work has been supported by the EU H2020 Programme under the SUNFISH project, grant agreement N.644666.

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

SAC 2017,April 03-07, 2017, Marrakech, Morocco

http://dx.doi.org/10.1145/3019612.3019719

2. BACKGROUND AND RELATED WORK

The resource augmentation of devices is being explored for several years now. Multiple approaches have emerged with different capabilities. All have one goal in common: Enable the usage of computational power of devices in the user's surroundings or of distant devices/clouds for the purpose of performance improvement or energy saving.

Pioneer work in the field of sharing resources among surrounding devices was first performed by Satyanarayanan [**?**] in 2001. He introduced surrogate computing concept, where surrogates in the user's proximity are used to outsource computations. With the rise of cloud computing, soon also the mobile cloud computing (MCC) paradigm emerged. MCC refers to the integration of cloud computing in the mobile environment to increase performance. Shiraz et al. [**?**] describe MCC as a new paradigm to extend the capabilities of devices with constrained resources. The next logic evolution step is introduced by Reiter and Zefferer [**?**] as Cloud-based Mobile Augmentation (CMA) and by Sanaei et al. [**?**] as Hybrid Mobile Cloud Computing (HMCC). These concepts make use of both former approaches: proximate devices and distant cloud resources.

Most existing solutions use the MCC approach. Cuervo et al. [**?**] developed MAUI as one of the first solutions. Their main goal is to extend battery lifetime by identifying and offloading computational expensive parts, using a developer assisted approach. In contrast to MAUI, which operates on method level granularity, the CloneCloud framework introduced by Chun et al. [**?**] provides a virtual clone of the user's mobile system in the cloud. Computational expensive parts are automatically identified and are executed in the cloud. This approach takes the burden off the developer, but these approaches require a lot of effort to keep the application state on the user's device and the clone synchronized. This issue is tackled by Kosta et al. [**?**] in their ThinkAir framework, which combines the approaches used by MAUI and CloneCloud. ThinkAir induces an additional compilation step where the code for the remote execution is generated. Reiter and Zefferer [**?**] introduce POWER, a HMCC framework as the only approach which also tackles the issue of interoperability, by focusing on cross-platform development frameworks.

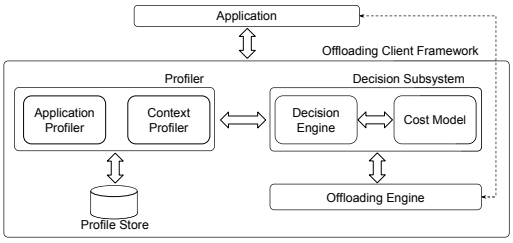

Figure 1: Generalized framework architecture

A consolidated architecture of the existing frameworks is shown in Figure 1. Frameworks generally make a clear distinction between four components: The *Profiler* provides performance values of the application execution and the application context, like the available bandwidth, the current latency to remote computing units, or the current battery status. The goal of the *Decision Subsystem* is to arrive at a final decision whether a method call should be offloaded or executed locally. For this process, the profiler values are used and applied to the *Cost Model*, which determines if offloading is beneficial. The focus of cost models varies from achieving high energy savings, to increasing the overall performance of the executed application. The *Offloading Engine* is responsible for the technical offloading process.

The gap in this architecture is that data- and application security is not considered by any of the components. The goal of this work is to respect data- and application security already on an architectural level and give the data- and process owners full control of the process flows.

3. A DATA SECURITY FOCUSED HMCC ARCHITECTURE

From a data owner's perspective the optimal situation is to completely decouple the decision on where to execute an offloaded task, from the actual application logic. This goal is similar to the goal of decoupling access control decisions from the application logic, which is the prime use case of the *eXtensible Access Control Markup Language* (XACML). XACML enables to write policies, evaluated on a set of provided attributes. Beside a XML-based access control policy language, the XACML specification provides best practices for the enforcement of the defined policies. Without going into the very details, the following components are introduced: The *Policy Enforcement Point* (PEP) is the connection point for applications, retrieves decisions from the *Policy Decision Point* (PDP), and enforces these decisions. The PDP receives policy decision requests from the PEP, with relevant attributes attached, and gathers policies matching the attributes from the *Policy Administration Point* (PAP). It then evaluates the policies on the received decision request and gathers any missing attributes from the *Policy Information Point* (PIP) if necessary. The result can be linked to *obligations* which need to be fulfilled by the PEP before the result is passed to the application.

3.1 Requirements

XACML infrastructures are already deployed in many corporations for access control to certain services and applications. As Cuervo et al. [**?**] show, the latency to the remote server has a significant impact on the energy consumption of mobile devices and on the performance of offloaded parts of an application in HMCC scenarios. Therefore, to make use of the full potentials of HMCC, 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. To achieve this goal, and to be able to successfully and effectively transform HMCC operations to XACML, the following requirements need to be fulfilled by the policy infrastructure beyond access control:

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

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.

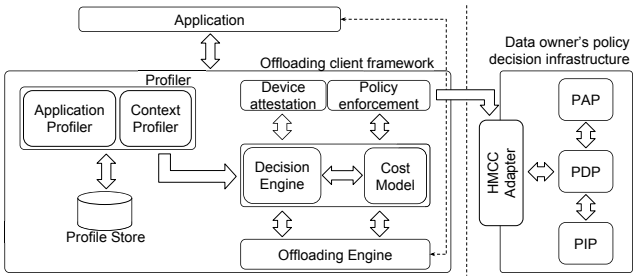

Figure 2: Security aware HMCC architecture

3. Caching of policy decisions needs to be enabled by using a time-based validity constraint.

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

3.2 Architecture

Our novel HMCC architecture extensions are highlighted in Figure 2 and take the defined requirements from Section 3.1, as well as an integration into existing policy infrastructures into account. The figure shows three major extensions. The decision subsystem is supported by a policy enforcement component, which corresponds to the PEP in a XACML setup. Furthermore, a device attestation component is introduced that is capable of assessing if another device fulfills security requirements defined by the policies. The data owner's already existing policy decision infrastructure is included into the HMCC framework by introducing the HMCC adapter component. The HMCC adapter transforms requests received from the client side policy enforcement component, to a XACML representation and returns the verbose XACML responses in an appropriate format. PDP responses are in XML format probably with heavy XML signatures attached. Processing XML is CPU and memory intensive [**?**, **?**] compared to JSON. The HMCC adapter acts as a converter from the proposed JSON representation, especially targeting the HMCC use case to XACML and vice-versa for the response.

A sample HMCC adapter request, targeting frameworks using method-level approaches is shown in Listing 1. This request is generated by the policy enforcement component in the client framework during the decision process if no matching cached decision response is available.

Listing 1: HMCC adapter request

```
{"id": "unique-id",
 "package-name": "package-name",
 "application-version": "0.0.1",
 "method-name": "method-name",
 "parameter-values": {
  "arg1" : 10, ...} ...}
```

The request contains parameters to uniquely identify the request and to uniquely identify the current part of the application considered for offloading with relevant associated input data. The structure of this request can be freely adapted, depending on the actual used framework and offloading approach. The decision, if a parameter value is included in the policy decision request is up to the offloading framework, which needs to provide measures to identify relevant parameters.

The response received from the HMCC adapter after converting the XACML decision response to the JSON format is illustrated in Listing 2. It contains information on where an execution is allowed in general terms: *specific*, *no-requirement*, *trusted*. For strict environments, where the policy decision infrastructure needs explicit control where an execution takes place, the *specific* execution mode can be used. Execution is only allowed on computing nodes listed in the response, identified with a matching certificate. If the currently considered application part and input data are uncritical, the *no-requirement* execution mode can be used. In this case, the selection of an appropriate node is up to the client-side offloading framework. Using the *trusted* execution mode, the decision infrastructure relies on device attestation services to determine if a remote device is trusted and can be used to offload critical operations. In contrast to the first option, this method adds flexibility to the system because (a) the decision infrastructure does no need to be aware of all computing nodes, and (b) the client-side offloading framework can dynamically react to changing system conditions and can use the best performing node in its surrounding without issuing policy decision requests multiple times. This targets the second requirement defined in Section 3.1.

Listing 2: HMCC adapter response

```
{"id": "unique-id",
 "execution-mode": "trusted",
 "nodes": [
    { "endpoint": "client identifier",
      "identity": "certificate fingerprint"},
    ...],
 "obligations": [...],
 "validity": {
  "valid-for": <time-based validity constraint>,
       "constraints": [...]}}
```

The HMCC adapter's response also contains validity constraints targeting time-based validity and validity restrictions focused on the input data. Time-based validity is realized in a straight forward way, by providing a validity timespan. Validity constraints are realized using a predicate language for JSON, based on the draft of JSON-predicate[2]. Using this approach, the creators of policies can express client-side validity terms. Under these terms, the policy decision does not need to be retrieved again. These measures target the first and third requirement as defined in Section 3.1 and enable an effective client-side caching of policy decisions. With policies and constraints matching wide ranges of processes and input data, the number of required requests to the policy decision infrastructure is minimized.

To target the fourth requirement from Section 3.1, policy decisions are not transmitted in plain but are signed using JSON Web Token (JWT) [**?**] technology. In contrast to XML signatures, JWT offers a lightweight signature approach with little overhead, best suited for resource-constrained devices.

Another advanced feature is the direct inclusion of XACML obligations. This can be used to e.g. offload an uncritical

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

operation on sensitive data to untrusted nodes, by applying encryption or data masking on the sensitive data before performing the offloading operation. This goes beyond the scope of this paper, therefore it is not considered in detail here.

4. IMPLEMENTATION

We reviewed existing frameworks to find a candidate that fits our requirements best and can be used to prove the feasibility of our proposed extensions. We think that POWER [**?**] is a suitable and future-proof candidate due to its modular approach.

In the POWER framework, developers apply annotations on source code level, to define which parts of the application should be considered for offloading by the offloading framework. This approach is extended to additionally provide information on relevant input data. The enhanced annotation is shown in Listing 3. It includes a listing of parameters to be transmitted to the PDP. Based on these attributes, policy administrators can create targeted policies and JSON predicates. Furthermore, the developer specifies which (predefined) decision point the framework should use.

Listing 3: Policy-aware annotation

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

As discussed in Section 3, three modes of execution are introduced, as a result to the policy decision request. The *specific* and *no-requirement* execution modes are clear and self-contained, whereas the *trusted* execution mode requires additional services to attest the trust state of connected computing units. We looked at different options to assess the trust-state of remote server/desktop environments and remote mobile environments and are presenting three approaches here. We focused on lightweight approaches, suitable for our dynamic use case:

Intel Software Guard Extension (SGX): Intel introduced SGX recently, a CPU extension which enables the creation and execution of protected software containers, called enclaves. Enclaves are protected by hardware-enforced access control policies and directly operate on the hardware, without an operating system layer. Baumann et al. [**?**] introduce a system called *Haven*. With Haven, they achieved to run applications like Microsoft SQL Server and the Apache Webserver in an Intel SGX enclave. Basically, they are running a thin operating system layer in each enclave and launch the unmodified applications on this layer. They reached the limits of SGX and propose changes for the coming iterations of Intel SGX. Furthermore, this approach requires changes to the Windows kernel, therefore we were not able to use this technology for demonstration purpose yet, but it seems to be the most promising candidate for cloud, server and desktop attestation and secure execution.

Google SafetyNet API: Mobile devices generally offer good isolation between apps. If security mechanisms of recent mobile devices are used, apps can protect their data in a way that no other entities can access the data. This is valid as long as the device is not rooted. No security guarantees can be made for rooted devices. The SafetyNet API[3] is a service included in the Google Play services, thus available on all recent Android releases. A positive attestation result strongly indicates that no one tampered with the device, the device is not rooted and app isolation can be guaranteed. To retrieve an attestation result, the client device contacts the SafetyNet server with a unique nonce. The service returns a signed statement, representing the device's state. A separate verification service is available to check the validity of acquired assertions. For integration in the POWER framework, a POWER application needs to run on the device, which can be triggered remotely and returns the attestation result to the initiating device.

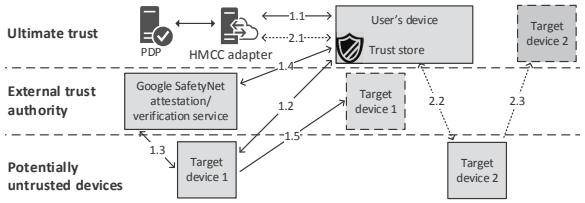

Figure 3: Trust propagation through the system

Certificate-based Approach: This approach is comparable to the former SafetyNet approach but is targeted at corporations. The SafetyNet approach is targeted at all Android devices, regardless if they are managed by a corporation, or by the end-user. Corporations often centrally manage their devices using *Mobile Device Management* (MDM) software, tightly integrated into their enterprise infrastructure. MDM solutions provide an even more fine-grained control of what is allowed on a specific device. Using this technology, corporations are able to provide secured devices with pre-installed certificates and the relevant application to participate in the HMCC network. This approach offers comparable security guarantees to the SafetyNet approach but enables a more fine-grained and corporation controlled setup. The integration in the POWER framework is done via the *specific* execution mode. A corporation's policy decision infrastructure is aware of the deployed certificates on the mobile devices, therefore it can provide the trusted link to the used certificates in the decision response.

5. EVALUATIONS

In this chapter, we evaluate two aspects of our proposed architecture and developed implementation. The trust model in Section 5.1 illustrates the propagation of trust through the system and in Section 5.2 we evaluate the performance trade-off induced by our architecture.

5.1 Trust Propagation Model

The trust propagation refers to the assigned trust levels of each component and their interactions with other components. Through the interaction with other components, and with attestation mechanisms and protocols in place, components with an undetermined trust level can be evaluated.

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

The trust propagation model of this work is illustrated in Figure 3. It consists of three different trust levels: *ultimate trust* for the core components and the user's device, *external trust authority* for external attestation service (e.g. Google SafetyNet), and *potentially untrusted devices* for all other devices which are untrusted or where the trust level is not clear yet.

Figure 3 contains the relevant interactions of two described trust assessment use-cases indicated by **1.x** (trust assessment using Google's SafetyNet approach) and **2.x** (trust assessment using certificates).

In the first use case, the user's device connects to the HMCC adapter in the first step. The HMCC adapter authenticates using HTTPS (or similar technologies) and verifies if the host is trusted by verifying the HMCC adapter's certificate against its trust store. It then acquires a HMCC adapter signed decision result. The signature is checked against an embedded set of certificates in the offloading framework on the user's device. At this point, the device can be sure that it is talking to a valid policy decision environment. In the next trust relevant steps **1.2** and **1.3** the user's device generates a nonce and requests a SafetyNet attestation from a potentially untrusted device. The user's device then checks the embedded nonce and requests a cryptographic verification from the SafetyNet verification service. Additionally, the user's device can check the embedded data in the attestation response, like the issuing package, and the hash of the APK requesting the attestation. These are values known to the policy decision infrastructure and could be transmitted to the user's device in step **1.2**. If all these checks succeed and the attestation response indicates a non-tampered target device, the trust level of the target device is increased to the same level as the external attestation service. Depending on the policies, this level might be equal to the ultimate trust level.

For the second use case, the first step is performed analogously, with the addition that the HMCC adapter transmits trusted node identifiers with associated certificates. This is targeted at corporation-managed devices, equipped with a certificate used for authentication. In this use case, the user's device establishes a connection to the target device, through means provided by the offloading framework and performs an authentication. In the POWER framework, this is realized by establishing a TLS connection with client authentication. If the certificate used during authentication matches the certificate received from the policy decision infrastructure, the device's trust level is increased to ultimate trust. This approach is comparable to certificate pinning, but the pinned certificates are not embedded in the application. They are received from the trusted policy decision infrastructure.

5.2 Performance Evaluations

To evaluate the performance trade-off of the added policy evaluation capabilities, we performed two policy-related evaluations: First, we measure the time it takes to evaluate policies when requests are dispatched to the policy decision infrastructure. Furthermore, we capture the required time to evaluate cached policies, where no request to the policy decision infrastructure is required. The second evaluation is based on the previous evaluations performed with the POWER [**?**, **?**] framework.

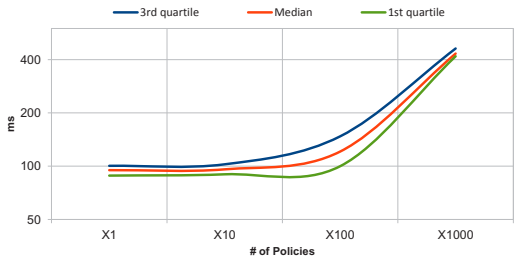

Figure 4: Policy evaluation performance

The policy evaluation benchmark is executed on an Amazon EC2 m4.xlarge2 instance running the 64-bit version of Ubuntu 14.04 LTS. The use of an EC2 instance enables a direct comparison of the execution times with other architectures or other approaches. The test environment is a standard XACML environment as already illustrated in Figure 2, based on the implementation of Apache OpenAz.

For the evaluation, we populated the PAP with 1, 10, 100 and 1000 policies and generated requests, where each of these requests was sent 100-times to the evaluation infrastructure. We measured the time it takes for the evaluation infrastructure to come to a decision and the time it takes for the HMCC adapter to convert the request. The results in Figure 4 show the mean request-evaluation time as well as the interquartile range (IQR).

For 100 policies in the policy store, it takes about 96ms to evaluate a decision request, and transform and attach a signature by the HMCC adapter. These are good performance values, especially considering the complex workflow of the decision infrastructure. For HMCC applications without policy caching this adds 96ms to every request. This would render some applications unusable. Therefore, we also introduced the policy caching approach, where only a single call to the decision infrastructure is required and many calls can be handled locally, without contacting the policy decision infrastructure.

To evaluate the policy caching performance, we measured the average time required to retrieve and match a cached policy. The average evaluation time of a single policy is calculated based on 100 executions of the evaluation procedure. Our test-setup considered policy decisions with constraints, and without constraints. Finding policy decisions with constraints, is computationally more intensive because matching requires to also evaluate the constraints on the provided input data. As test systems we used a Motorola G2 with Android 5.1, and a Lenovo X230 running on Windows 10 with Google Chrome 53. We could not determine significant performance differences for different mobile devices or desktop systems. The results in Table 1 show that on mobile devices, the evaluation of cached policies without constraints is about 56-times faster, and with constraints about 24-times faster, than contacting the policy evaluation infrastructure.

The goal of the second evaluation is to determine the per-

	Android	Desktop
Without constraints	1.71ms	0.32ms
With constraints	3.95ms	1.43ms

Table 1: Average evaluation time of a single cached policy

formance impact of our security and privacy-focused approach. To highlight the impact and to get comparable results with previous versions of POWER, but also with other mobile application offloading-frameworks, we re-evaluated the performance benchmark on general programming constructs (GPC) as performed by Reiter and Zefferer [**?**]. The goal of the evaluation is to show at which complexity *Fibonacci*, *Nestedloop* and *SHA1* benchmarks benefits from offloading. As described in Section 4 we extended the testcases to make use of our policy approach, also utilizing the described constraint approach. The PAP was populated with 100 not matching policies and a single policy matching each of the test cases. The evaluation results are shown in Table 2. The results do not contain the time required for the initial population of the policy cache. The original results can be seen in the second column, whereas the new results, gathered using the policy-enabled approach, can be seen in the third column. The time required to evaluate a single cached policy decision is only a fraction of the time required for the actual task. Still, for the improved security and privacy awareness of the framework, small trade-offs in performance need to be accepted. To take the example of SHA1: Using the standard POWER framework, a complexity of 41 is required before a performance improvement through mobile application offloading manifests. The policy enabled approach requires a complexity of 44 to benefit from application offloading. In conclusion, all benchmarks show increases in the minimal required complexity. Still, they are minimized due to advanced policy caching techniques and enable an efficient usage of mobile application frameworks and protect the data owner's assets.

Benchmark	Original complexity	Complexity with policy approach	Data Tx (bytes)	Data Rx (bytes)
Fibonacci	23	25	~450	~80
Nestedloop	16	17	~450	~80
SHA1	41	44	~450	~100

Table 2: GPC benchmark results

6. CONCLUSIONS AND FUTURE WORK

In this paper we proposed an extension to device-to-device offloading frameworks, to enable secure policy-based approaches. Our proposed architectural changes are based on existing frameworks, and therefore can be widely adopted by different frameworks. The main goal of the enhancements is to give data owners full control of their data but still, benefit from the increased performance of device-to-device offloading approaches. To prove the feasibility of our approach we extended the POWER framework and applied the proposed enhancements. The evaluation shows that the impact of the policy decision infrastructure is reduced to a minimum, due to the used caching technique. We think that this is the next step in the evolution of HMCC systems.

In the future, we will mainly concentrate on improving the policy caching mechanism and enable more complex constraints to reduce the required number of requests to the policy decision infrastructure even further.

7. REFERENCES

[1] A. Baumann, M. Peinado, and G. Hunt. Shielding Applications from an Untrusted Cloud with Haven. *Proceedings of the 11th USENIX Conference on Operating Systems Design and Implementation*, pages 267–283, 2014.

[2] B. Chun, S. Ihm, P. Maniatis, M. Naik, and A. Patti. CloneCloud: Elastic Execution Between Mobile Device and Cloud. In *Proceedings of the sixth conference on Computer systems*, pages 301–314, 2011.

[3] E. Cuervo, A. Balasubramanian, D. Cho, A. Wolman, S. Saroiu, R. Chandra, and B. Paramvir. MAUI : Making Smartphones Last Longer with Code Offload. In *Proceedings of the 8th international conference on Mobile systems, applications, and services*, volume 17, pages 49–62, 2010.

[4] M. Jones, J. Bradley, and N. Sakimura. JSON Web Token (JWT). RFC 7519, RFC Editor, May 2015. http://www.rfc-editor.org/rfc/rfc7519.txt.

[5] S. Kosta, A. Aucinas, P. Hui, R. Mortier, and X. Zhang. ThinkAir: Dynamic resource allocation and parallel execution in the cloud for mobile code offloading. In *Proceedings of 2012 IEEE INFOCOM*, pages 945–953, 2012.

[6] B. Lin, Y. Chen, X. Chen, and Y. Yu. Comparison between JSON and XML in Applications Based on AJAX. *Proceedings - 2012 International Conference on Computer Science and Service System*, pages 1174–1177, 2012.

[7] N. Nurseitov, M. Paulson, R. Reynolds, and C. Izurieta. Comparison of JSON and XML Data Interchange Formats: A Case Study. *Caine*, pages 157–162, 2009.

[8] A. Reiter and T. Zefferer. POWER : A Cloud-Based Mobile Augmentation Approach for Web- and Cross-Platform Applications. In *Proceedings of 2015 IEEE CloudNet*, pages 226–231, 2015.

[9] A. Reiter and T. Zefferer. Flexible and Secure Resource Sharing for Mobile Augmentation Systems. *2016 4th IEEE International Conference on Mobile Cloud Computing, Services, and Engineering (MobileCloud)*, pages 31–40, 2016.

[10] Z. Sanaei, S. Abolfazli, A. Gani, and M. Chen. HMCC: A Hybrid Mobile Cloud Computing Framework Exploiting Heterogeneous Resources. In *2015 3rd IEEE International Conference on Mobile Cloud Computing, Services, and Engineering*, pages 157–162, 2015.

[11] M. Satyanarayanan. Pervasice Computing: Vision and Challenges. *IEEE Personal Communications*, pages 10–17, 2001.

[12] M. Shiraz, A. Gani, R. H. Khokhar, and R. Buyya. A Review on Distributed Application Processing Frameworks in Smart Mobile Devices for Mobile Cloud Computing. *IEEE Communications Surveys & Tutorials*, 15(3):1294–1313, 2013.