Fachhochschule Nordwestschweiz
Hochschule für Technik

Institut für Mobile und Verteilte Systeme

FOKUS REPORT

2014

45
9 771662 201005

Bildbetrachtung

Das diesjährige Titelbild zeigt einen LED-Cube, bestehend aus 512 roten, lichtemittierenden Dioden (LEDs). Die würfelförmig angeordneten LEDs können einzeln über Software angesteuert werden. Einige simple Animationen, aber auch interaktive Anwendungen wie ein 3D-Pong und das Darstellen von Twitter-Nachrichten sind von Christof Arnosti realisiert und am Tag der offenen Tür des neuen FHNW-Campus in Windisch präsentiert worden. Controller- und Twitter-Nachrichten werden über eine USB-Schnittstelle übertragen, alle Berechnungen finden jedoch direkt auf einem Arduino Nano mit ATmega Mikrocontroller im Würfel statt.

Jeweils acht LEDs in senkrechter Reihe sind an ihren negativen Polen miteinander verlötet und an einem der insgesamt 64 Ausgänge von gesamthaft vier LED-Controllern angeschlossen. Pro Würfelschicht sind die Pluspole von jeweils 64 LEDs miteinander verlötet und an einem Transistor angeschlossen, welcher für die notwendige Spannung pro Schicht sorgt. Sowohl die LED-Controller als auch die Transistoren werden über die verwendete Arduino-Plattform angesteuert.

Durch diesen Aufbau kann Multiplexing betrieben werden, d.h. für jedes auf dem Würfel dargestellte Muster werden für alle Schichten der Reihe nach alle auf einer Schicht benötigten LEDs auf den LED-Controllern aktiv geschaltet und anschliessend kurz mit Spannung versorgt. Durch die hohe Geschwindigkeit der Schichtwechsel (960 Hz) entsteht so die Illusion, dass einzelne LEDs durchgehend aktiv sind.

LEDs sind schon seit über fünfzig Jahren bekannt. Lange Zeit wurden sie nur zur Zustandsanzeige von elektronischen Geräten eingesetzt (z.B. grün für ein- und rot für ausgeschaltet). Durch ständige Verbesserung der Lichtausbeute trat die LED ihren Siegeszug gegenüber der Glühbirne an und wurde nach und nach auch in Automobilrückleuchten und in Verkehrsampeln verbaut.

Der Durchbruch der LEDs als Lichtquellen begann aber erst mit dem Aufkommen der weissen LEDs, genauer gesagt mit der Erfindung von effizienten blauen LEDs. Sie haben richtig gelesen: Die blauen LEDs waren für lange Zeit eine grosse physikalische bzw. technische Herausforderung, die erst Anfang der 90er Jahre einer Gruppe von japanischen Wissenschaftlern gelang. Weshalb diese Forscher erst heuer den Physiknobelpreis für ihre bahnbrechende Erfindung erhielten, ist eine andere Geschichte. Ähnlich wie bei Leuchtstoffröhren lässt sich das kurzwellige und höherenergetische Licht von blauen LEDs mit Hilfe von grünen und roten Leuchtstoffen in langwelligeres grünes bzw. rotes Licht umwandeln. Gemäss der

Inhalt

Impressum

Herausgeberin:
Fachhochschule Nordwestschweiz FHNW
Institut für Mobile und Verteilte Systeme
Bahnhofstrasse 6
CH-5210 Brugg-Windisch
www.fhnw.ch/technik/imvs
Tel +41 56 202 99 33

Kontakt: Prof. Dr. Jürg Luthiger
juerg.luthiger@fhnw.ch
Tel +41 56 202 78 23
Fax +41 56 462 44 15

Redaktion: Prof. Dr. Christoph Stamm
Layout: Claude Rubattel
Erscheinungsweise: jährlich
Druck: jobfactory Basel
Auflage: 150

ISSN 1662-2014 (Print)
ISSN 2296-4169 (Online)

additiven Farbmischung kombinieren sich dann blaues, grünes und rotes Licht zu weissem Licht.

Der Erfindung der blauen LED haben wir es somit zu verdanken, dass wir heute die wesentlich energieeffizienteren LEDs für die Beleuchtung einsetzen können. Erinnern Sie sich noch an die unzähligen Beiträge auf allen Medienkanälen, als die Glühlampen aus den Verkaufsregalen verbannt wurden? Und nun, als im Oktober dieses Jahres die Nobelpreise vergeben wurden, erwähnte das Schweizer Fernsehen den Physiknobelpreis mit einem einzigen Satz, während es den Nobelpreisen für Wirtschaft, Frieden und Literatur teilweise ganze Sendungen widmete. Ob einer der drei in den Medien gewürdigten Nobelpreise eine ähnliche grosse Praxisrelevanz wie die Erfindung der blauen LED hat, dürfen Sie selber beurteilen. Aber in den Medien und in grossen Teilen der Gesellschaft geht es oft nicht um Relevanz oder technische Bedeutung. Lieber werden die Menschen und ihr soziales Umfeld ins Blickfeld gerückt, nach dem Motto: Eine gute Geschichte ist eine *People Story*. Dass die technische Berichterstattung in den Medien eher abgenommen hat, obwohl die zunehmende Technologisierung unseres Alltags eine Mündigkeit in diesen Bereichen fordert, um an gesellschaftlich relevanten Diskussionen teilnehmen zu können, hängt wohl nicht zuletzt damit zusammen, dass die verantwortlichen Redaktionen selten technikaffin sind. Diese Abkehr von der technischen, wissenschaftlichen Berichterstattung in den grossen Medien bleibt jedoch nicht folgenlos: Immer mehr junge Menschen interessieren sich immer weniger für technische Berufe.

Das kürzlich erschienene MINT-Nachwuchsbarometer der Schweizerischen Akademie der Technischen Wissenschaften (SATW) geht der Frage „Wo bleiben unsere MINT-Fachkräfte?“ auf den Grund und listet eine Vielzahl von Erklärungsversuchen auf, weshalb die Technikbegeisterung bei vielen Kindern und Jugendlichen im Laufe der Schulzeit abnimmt und vor allem bei jungen Frauen bei der Berufswahl so tief gesunken ist, dass fast immer andere Berufswünsche überwiegen. Die Studie bringt auch klar zum Ausdruck, dass eine Förderung des naturwissenschaftlichen Interesses sich nicht automatisch positiv auf das Interesse an Informatik und Technik auswirkt, d.h. es bedarf einer speziellen und frühen Techniksozialisierung im Elternhaus und der Schule. Ein Fach Technik und Informatik in der obligatorischen Schule bedingt jedoch, dass anderweitig Stunden abgetreten werden müssen, was aber keinesfalls zu Lasten von Mathematik und Naturwissenschaften gehen darf. Im Vergleich zu anderen Ländern ist die Gewichtung von MINT-Fächern in den obligatorischen Schulen der Schweiz jetzt schon niedrig und sollte daher vor allem gestärkt werden. Diesbezüglich geht der neue Lehrplan 21 in die richtige Richtung, obwohl darin die Informatik der Medienbildung zur Seite gestellt wird, was einmal mehr deutlich werden lässt, wie verkehrt Informatik in der Gesellschaft wahrgenommen wird. Informatik ist nicht gleich *Chatten*, *Gamen*, *Liken*, *Posten*, *Surfen!*

Die Informatikbranche ist aber nicht ganz unschuldig an dieser verfehlten Wahrnehmung. Zu sehr hat sie sich im letzten Jahrzehnt damit beschäftigt, manuelle Werkzeuge und herkömmliche Prozesse zu digitalisieren und somit als Dienstleister aufzutreten, anstatt das Zepter selber in die Hand zu nehmen und neue vereinfachte, automatisierte Datenflüsse und Prozesse zur Verfügung zu stellen und zu etablieren. Kriegt beispielsweise ein Informatikdienstleister den Auftrag eine Software für ein Reisebüro zu entwickeln, dann entsteht daraus mit Sicherheit nicht *booking.com*.

Wer sich nur damit beschäftigt, die altbekannten Prozesse der Reiseplanung zu digitalisieren, der übersieht, dass Reisebüros überflüssig sind, wenn die Datenflüsse zwischen Fluggesellschaften, Mietwagengesellschaften, Hotels und Endkunden automatisiert werden. Das gleiche gilt auch für andere Dienstleistungsbranchen, die sich bis anhin hauptsächlich mit der Verwaltung und Vermittlung von Daten und Informationen beschäftigt haben. Hier soll und muss die Informatikbranche wieder mutiger werden und neue alternative Geschäftsprozesse den bisherigen zur Seite stellen. Mögen die besseren gewinnen!

Dazu braucht es in Zukunft genügend bestens ausgebildete Fachleute. Also zumindest relativ gesehen sollten sich junge Frauen und Männer vermehrt mit Informatik und Technik beschäftigten. Die bisherigen, kantigen Technikfächer abzuschleifen und sie damit für junge Frauen inspirierender zu machen, ist aber definitiv der falsche Ansatz. Vielmehr geht es darum, dass neue attraktive Berufsbilder und Abschlüsse für Frauen geschaffen werden, welche eine eigene Berechtigung haben und sich in der Wirtschaft etablieren können und müssen. Solche neuen Berufsbilder könnten zum Beispiel an den vielfältigen Nahtstellen zwischen Mensch und Maschine/Computer angesiedelt werden.

Erst wenn unsere Gesellschaft so viel Technikverständnis mit sich bringt, dass sie den Zusammenhang zwischen dem Nobelpreis für blaue LEDs und dem Stromsparen bei der Beleuchtung versteht, erst dann ist unsere Gesellschaft wirklich im 21. Jahrhundert angekommen und auf dem richtigen Weg, den demografischen Schwierigkeiten gelassener entgegen zu treten, im Wissen, dass noch sehr viel Automatisierungs- und Verbesserungspotential vorhanden ist, welches den weithin beklagten Fachkräftemangel als Chance und nicht als Bedrohung sieht.

Prof. Dr. Christoph Stamm

Qualitative Study of Successful Agile Software Development Projects

Various studies show that the agile method has become a mainstream methodology for software development. When agile pioneers introduced this approach, they executed very successful projects which lead to the enormous popularity of agile development. With becoming mainstream, less experienced teams started to apply the agile approaches and news about failed agile projects appeared. This raises the question, what it needs to conduct successful agile projects. In a qualitative study we asked IT companies about the essential success factors in their successful agile projects. We found that there was a strong focus on engineering and management best practices. We found that when these practices did not work, mature teams sensed that following a recipe is not sufficient, and they started adapting the agile process to their needs. Applying a sense-making methodology like the Cynefin framework, theoretically explains our observations in the study.

Martin Kropp, Andreas Meier | martin.kropp@fhnw.ch

In recent years, agile methods have become one of the predominant ways for software development as various studies show [1] [3]. This is not only because agile has become hype and any "state-of-the-art" IT company "just does it". The mentioned studies also clearly show improvements over traditional approaches, like faster time-to-market, significantly improved change management, and higher satisfaction with the overall process. However the studies also show, that a significant part of agile projects fail due to various reasons, though they apply most of the recommended good practices for agile development.

Thus the authors wanted to find out, what the essential factors for successful agile projects are. For this they conducted an interview study among several Swiss IT companies about their most successful agile projects. The goal was to get an understanding of what the essential criteria for successful agile projects are.

The first section provides an overview about related work. Then we present the qualitative study, with its central questions and the applied study method, followed by the major study results. In the next section we introduce the theory of Complex Adaptive Systems which helps to understand how successful agile software projects and teams function and work. After that we present and discuss the findings about the critical success factors from our interview study. The paper closes with a final conclusion and an outlook on future work.

Related Work

There are many studies, which show the distribution and success of agile methods for software development. The company Version One executes a yearly study about usage of agile methods and practices [1] in the United States. Reference [3] is a similar bi-yearly study focused on the usage of agile and plan-driven approaches in Switzerland. In [2] Kaltenecker et. al. present the results of an interview study with the focus on leadership in agile projects.

Already in 2001, A. Cockburn and J. Highsmith discussed the still neglected importance of the people factor when introducing agile methodologies [20]. Recent research [21] and [22] goes further and analyses the influence of organizational culture and agile methodology.

In [9] and [5] Snowden et. al. discuss the relation between complex systems and agile software development and present the *Cynefin* framework [4] for agile leadership.

Interview Study

The goal of our study was to show by example how successful agile projects are realized in various kinds of projects and different kinds of companies and branches. We conducted an interview study among eight Swiss IT companies that have adopted agile methods in their software development. The companies were asked to provide a written description in advance of their most successful agile project, including a reasoning why they value the described project as successful – from their point of view and from their customers' point of view.

We conducted semi-structured individual interviews and used a self-developed interview guide. We asked questions about all the three competence levels engineering, management and agile values as formulated in [7]. Since all interviews took place in the German speaking part of Switzerland, all interviews were conducted in German.

The interviewed companies operate nationally, internationally and globally. Table 1 gives an overview of the industry branches covered. The interviews were conducted with a total of nine participants (at one company, we had two interviewees, all male). The participants were most-

Branches	Number of Companies
Product Development	2
Public Service	1
Manufacturing	2
Insurance	1
IT Supplier	2

Table 1: Branches covered in study

ly project leaders, group leaders, or department leaders. The interview duration was between one and two hours. All interviews were conducted in German. All interviews were audio recorded and transcribed later. The transcription facilitated the evaluation of the interviews with the help of a statistical text-analysis tool.

Organizational Level of Agility

In this paper we use the terms *team*, *organization* and *company* to describe on which level Agility was introduced:

- *Team* refers to the software developers, testers, Scrum Masters, Product Owners etc., i.e. the people that are directly involved in the software development. All the teams in the study follow an agile methodology like Scrum.
- *Organization* refers to the team plus other people who are involved in the project, i.e. customers, end-users, and management sponsors. They work together with the team in an agile or non-agile environment. In the former case, the team supports the organization to become agile. In the latter case, the team provides an "interface" to facilitate the communication between the different environments.
- *Company* refers to the enterprise within which the software development project is executed. Currently, most companies are not agile. In our interviews, we had seven non-agile companies where at least one team or organization was agile.

In seven of the eight companies the agile approach was applied either for one or more organizations within the company or on the team level for the project teams. In these companies, agile methodologies were either introduced bottom-up or top-down. That means those teams where typically embedded in a classical, hierarchical organization within its own company. In one company, agile methodology was introduced in the whole company, i.e. was applied also on management level.

Success
1. What makes the project successful from your point of view?
2. What makes the project successful from your customer's point of view?
3. What are the reasons for the success from your point of view?
Engineering
4. How much do the engineering practices (according to [7]) contribute to the success?
5. Which engineering practices do you apply regularly?
Organization/Management
6. How much do the management practices (according to [7]) contribute to the success?
7. Which management practices do you apply regularly?
Culture & Values
8. How much do the cultural aspects / values (according to [7]) contribute to the success?
9. Which cultural aspects / values do you apply regularly?
Improvements
10. What would you change to make the project even more successful?

Table 2: Interview questions translated from German

Interview Questions

As preparation for the interview the interviewees were asked to send in a short description of the selected project, including some basic information about the project like duration, effort, and team size. Additionally, they had to answer the following two questions:

- What makes the project successful from your point of view?
- What makes the project successful from your customer's point of view?

In the one-hour interview (in average), the interviewee had to provide further information about

	P1	P2	P3	P4	P5	P6	P7	P8
Type of Project	Product	In-house	Product	Product	Product	In-House	In-House	Product
Company	IT Solution	Public service	Manu-facturer	Manu-facturer	IT Service Provider	Insurance	IT Solution	Manu-facturer
Method	Scrum	Scrum	Scrum	Scrumban	Scrum	Scrum	Scrum	Scrum
Size	24 PM	30 PM	30 PM	100 PM	30 PM	12 PM	72 PM	> 120 PM
Duration	3 M	10 M	12 M	9 M	10 M	8 M	24 M	>
Team	5 P	10 P	5 P	3 x 4 P	6 P	8 P	5 P	5 (current)

Table 3: Project figures

the company and the project (industry sector, company size, in-house/contract work/product development, main project technology). The questions are listed in Table 2.

Results of the qualitative study

Table 3 gives an overview of the basic project figures. The projects ranged from small projects to intermediate sized projects, and covered in-house projects, embedded software projects and product development. The team size ranged from 5 to 12 persons, the latter being organized as a scrum-of-scrum team.

Success in Agile Projects

The question about why the selected project was considered successful was answered in very many different ways. Among the most popular answers were: very good communication in the team; continuous delivery; delivery on time; very few bugs; satisfied customers; no overtime. When asked for the reasons for the success, the following main aspects were mentioned: all team members were committed; continuous and extensive testing; requirements were very clear; very close communication with and intensive feedback from customers; team workshops for team building.

Engineering Practices for Success

Many different engineering practices are applied by these organizations. Among the most important practices mentioned by almost all teams are continuous testing, at least on unit level (only few apply TDD); continuous integration; and continuous quality control. Some teams use Pair Programming, and some Clean Coding.

Organization Practices for Success

The success criteria on management level which were mentioned most often were the short iterations (typically two weeks) and the self-organization of teams.. More important issues mentioned by others were User Story Grooming during Sprint; close communication with externals; open office. Management support was also argued to be very important by one company. For remote teams daily meetings with visual communication tools like Skype was very important in one company.

In the eight observed projects, many different practices and paradigms were used. Their usage depended on the size and domain of the project but also on the culture of the respective company. We found a small but powerful set of underlying characteristics common in all agile projects. Before summarizing these key characteristics of successful agile projects, we will take a closer look at the theory of complex systems. With the insight of complexity theory, these key characteristics can be better understood and an indication on how and when to apply them in other projects can be deduced.

Complex Adaptive Systems and Agile Competences

In recent years complexity theory has been widely discussed in the scientific community. In this paper we argue that parts of a software development project should be treated as a complex system. More specifically we will point out that some parts of a project are in the *complex domain* and other parts are in the *ordered domain*. It is important to note that a typical software development project is a multi-domain problem. We elaborate what the consequences are and why that matters.

There are different definitions of complex systems. In this paper we use the definition of constraints. Basically there are three different types or ontologies: *chaotic* systems, *complex* systems and *ordered* systems. A system consists of *agents* and the interaction between these agents. Agents interact within the system or on the system. Examples of agents in software development are: the project goal, the project team, customers, the company, the culture within the company, management, competitors, technology, market place, etc. It is important to note that agents are not usually single individuals.

In a chaotic system, the agents are not constrained and thus independent of each other. In an ordered system, the system constrains the agents. These are the two extremes of the spectrum. In between there is the complex system. Such a system slightly constrains the agents. The agents modify the system by their interaction with it and among each other.

A *Complex Adaptive System* (CAS) is a special case of a complex system. Here the constraints and agents are co-evolving. Examples of CAS are the Internet, cyberspace, stock markets, manufacturing businesses, ant colonies, and any human social group-based endeavor. And a software development project is also regarded as a complex adaptive system [9].

Listed below are some of the characteristics of complex adaptive systems [9]:

- A CAS is highly sensitive to small changes. There is the danger that weak signals are easily missed or even dismissed. Small changes can have unpredictable consequences. This is also known as the butterfly effect.
- There is no (or very little) causality and therefore hindsight does not lead to foresight. In other words, a CAS is not causal but dispositional.
- Retrospective coherence, i.e. the fact that a system worked in a certain way in the past does not mean it will work the same way in the future.
- Proximity and connectivity are key

- There are dangers of confusing correlation with causation and simulation with prediction
- "Need to keep your options open". Premature convergence is the main danger of complexity, i.e. converting too quickly to a solution and not to keep multiple possibilities open.

If we look at agile software development projects as complex adaptive systems, there are a number of consequences that follow [9]:

- CAS cannot be reset. Wherever you are, is where you are, e.g. it is not possible to reset a sprint and start over.
- Relationships between agents are more important than the agents themselves. It is more effective to improve the relationships than trying to change every agent. Especially when the agents are groups of individuals.
- Management of ordered and complex systems or domains is important. The team has to know, what kind of domain the project is in and how to respond accordingly (see Cynefin framework below)
- Praxis[1] makes perfect. Agile teams value both theory and practice, e.g. they do not only apply Scrum in practice but they also know the theory of complex systems behind it.

Cynefin

Cynefin [4] is a decision-making framework that recognizes the causal differences that exist between chaotic, complex and ordered systems. It is valuable because it proposes different approaches to decision-making in complex social environments like in software projects.

The name Cynefin, pronounced ku-*nev*-in, is a Welsh word that signifies the multiple factors in our environment and our experience that influence us in ways we can never understand. "*The name seeks to remind us that all human interactions are strongly influenced and frequently determined by the patterns of our multiple experiences, both through the direct influence of personal experience and through collective experience expressed as stories.*" [5]

The Cynefin framework is also a sense-making framework. It is used to help define the system we have to deal with, i.e. lays the problem at hand in the ordered or un-ordered domain. This is important because problems in the ordered domain require very different strategies than the ones in the unordered domain.

Its value is not so much in logical arguments or empirical verifications as in its effect on the sense-making and decision-making capabilities of those who use it. The Cynefin framework has been used for knowledge management, project management, IT-design, strategy making etc. Its purpose is to give decision makers (e.g. the team members, agile leaders, management, etc.) powerful new constructs that they can use to make sense of a wide range of unspecified problems. It also helps people to break out of old ways of thinking and to consider intractable problems in new ways [4], [5].

1 There are different definitions of ‚praxis'. Here ‚praxis' is defined as the combination of theory and practice.

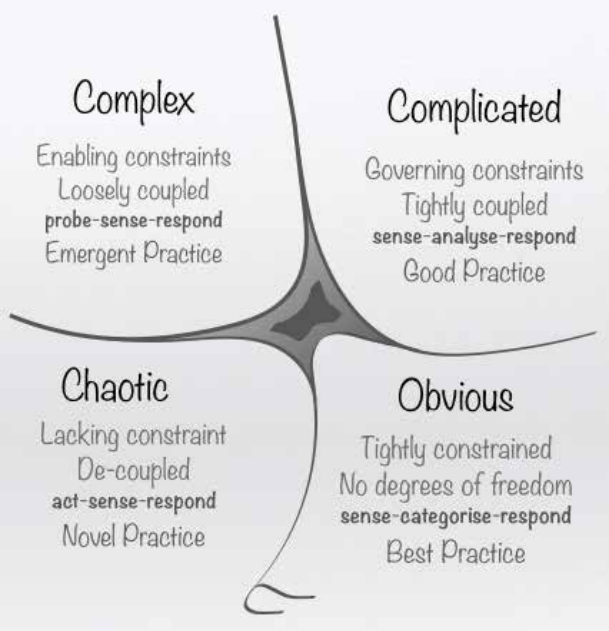

Figure 1: Cynefin domains (Source [11])

The Cynefin framework does not automatically provide the right answer, but it will help the agile team members to use their skills and experience to look for it in the right place.

The five domains of the Cynefin framework

As can be seen in Fig. 1, the Cynefin framework has five domains, four of which are named, and a fifth, central area, which is the domain of Disorder. The right-hand domains (Obvious, Complicated) are those of order, and the left-hand domains (Complex, Chaotic) are those of unorder.

- *Obvious Systems:* In the obvious (aka simple) domain, cause and effect relationships are obvious and predictable in advance. The causality is self-evident or obvious to any reasonable person. In this domain we apply *best practices*, i.e. established examples of what works in a particular context. The approach is to *sense, categorize* and *respond*.
- *Complicated Systems:* In the complicated domain we have an ordered system where cause and effect relationships exist but they are not self-evident. There is a right answer, however, the answer is not self-evident and requires analysis and/or the application of expert knowledge. With the right expertise, there can be several different ways of doing things in this domain. Applying *good practices*, i.e. a range of examples of what works well in a given context, works well in complicated systems

provided we have the right expertise. The approach is to *sense, analyze* and *respond*.

- *Complex Systems:* In the complex domain we have an un-ordered system where the relationship between cause and effect are only obvious in hindsight. The causality can only be perceived in retrospect and the results are unpredictable. In this domain, we need to create *safe to fail experiments.* And we do not attempt to create fail-safe designs. We cannot solve complex problems with best or good practices alone. While conducting safe to fail experiments, the key is to *dampen* the parts that fail and *amplify* the parts that succeed. In this domain we get emergent order and practice that is often unique. The approach is to *probe, sense,* and *respond*. In this domain we apply *emergent practices*, i.e. new practice and some combination of best and good practices.
- *Chaotic Systems:* In the chaotic domain, no cause and effect relationships can be determined. The approach is to *act, sense,* and *respond*. In order to effectively understand and function in a chaotic system, we must act very quickly to either innovate or stabilize the system and therefore learn from it. In chaotic systems we apply *novel practices*.

Depending on the system or ontology that applies to the situation, we should think and analyze accordingly. One size does not fit all!

- *Disorder:* The central space in Fig. 1 is key. It is called Disorder, the state of not knowing which system we are in. The danger of being in disorder, is that we tend to interpret and assess the situation according to the system we are most comfortable with, i.e. we compete to interpret this domain according to our preference for action. For instance, those people most comfortable with order seek to enforce rules, and experts seek to conduct research. This can be dangerous. Unfortunately, it is not uncommon to be in the domain of Disorder.

Figure 2: Pyramid of Agile Competences

Agile Competences

Developers in agile software development should possess a set of different competences. These competences can be divided into three categories as shown in the Agile Pyramid of Competences (Fig. 2), which was first presented in [7]:

- Engineering practices
- Management practices
- Agile values

The pyramid initially came from an educational point of view but proved to be useful as a guideline for the kind of competences an agile developer should have, and thus also for the questionnaire and interviews in the study. The analogy of the pyramid visualizes the decreasing number of required skills or competencies for an agile software developer from bottom to top. On the other hand, it reflects the increasing difficulty to learn or acquire these competences. Engineering practices can be taught very well in the classroom or be learned by the individuals on the job at their own pace. Management practices are, in our experience, best learnt through projects in teams. The most difficult competences to teach or learn are the agile values on top of the pyramid, since they often require a change in the attitude of the individual.

With the pyramid we also visualize the order in which the different practice levels should be acquired. Learning the management skills before a developer has a good working knowledge of the underlying engineering skills is like trying to build a pyramid from top to bottom. It might be possible, but at a very high cost (unfortunately, many companies are using this approach: They send their developers to a two day Scrum Master course and expect them to be agile after that). The same is valid for the agile values. Developers usually hear about the values in an Agile or Scrum course. But to properly understand and act accordingly requires a lot of experience and time.

Relationship between CAS and Agile Competencies

We found that the Theory of Complex Adaptive Systems and the Pyramid of Agile Competences are related. The practices at the bottom of the pyramid are typically in the ordered domain, i.e. obvious or complicated (Cynefin framework). There is a lot of expert knowledge necessary to master the engineering practices and that is the reason why it takes a long time for the students or developers to acquire it.

For instance, if the *good practice* of "Automated Unit Testing" is properly applied, there is a relation between cause and effect, e.g. the more (good) tests you write, the higher the quality of the software gets. And the software is also easier to refactor. It is well known that many of the XP-practices like "Automated Unit Testing" and "Refactoring" positively reinforce themselves.

This causality has been well described in [8]. In general, most engineering practices are in the complicated domain where expert knowledge is required and *good practices* can be applied. However, one might argue that some practices are in the simple space. "Clean code" for instance, may be considered an example of a practice in the simple domain.

Moving up the pyramid we get to the management practices. On this level there are more people involved and things start getting more complex. Communication outside of the team within the organization or with customers becomes important. Customer relationships are difficult to manage and so there is a transition between the ordered and the complex domain. There are no simple recipes to follow. Strategies that worked for the last project might be useless or even dangerous for the current project. When the agile (project) leaders are aware of this, they can take appropriate measures and apply *emergent practice* in order to adequately keep the project on track.

On top of the pyramid are the agile values or culture. Cultural aspects are definitely out of the ordered domain and belong to the complex domain. In this domain, the agile leaders and their teams are often learning by doing. Or more precisely, they must dampen the parts that fail and amplify the parts that succeed.

In [7] the authors argued that teaching and learning the agile values on top of the pyramid is much more difficult than the practices at the bottom. With the theoretical insight of the Cynefin framework, it is not difficult to understand why. Moving up the pyramid means transitioning from the ordered to the complex domain (or if you are unlucky: to chaotic domain).

Findings

It should be clear by now, that a list of good or best practices, which the team carefully follows, is not sufficient. It takes more for a project to become a success. In this chapter we present such a list of practices and additionally an overview of similarities or patterns, which we identified in the examined projects. These patterns appear with different characteristics. In some projects the patterns were strongly formed, in others less. These patterns are not meant as a recipe to follow. They are merely a collection of *emergent practices*. As we have seen in complex adaptive systems, we need to create *safe to fail experiments* and act depending of the outcomes. The observed emergent practices may be useful as a starting point to create safe to fail experiments in other projects.

We present the findings following the three levels of the Pyramid of Agile Competencies. We start at the bottom with the engineering practices.

Engineering practices

In the interviews, all the teams reported that they emphasize the engineering practices. These practices are seen as very important and as a kind of foundation, which ensures that software can be developed in short iterations with the required quality. There is a constant process of improvement and refining of these practices. Depending on the maturity of the team and organization, this is triggered by the agile champion (see below), the team members or, in the case of an agile company, even by the management. This can happen ad hoc or formalized. Engineering practices are mostly well-known good or best practices. Many of them have been popularized in eXtreme Programming [8]. A quantitative overview can be found in [1], [3].

Refactoring, automated tests, continuous integration and deployment, pair programming, test-driven development [18], etc. are the tools of the trade [19]. Testing, continuous integration and clean coding was mentioned as core practices in most organizations.

1. *Testing:* In almost all the organizations automated testing on unit level is well established and is seen as an absolute must for providing a good software quality. More mature organizations also apply automated testing on acceptance testing level using new approaches like Automated Acceptance Testing (ATDD) and Behavior Driven Development (BDD).
2. *Continuous Integration:* Continuous integration is seen as an absolute must for being able to deliver software with high frequency. Thus all organizations have established an automated build and test environment which provides immediate feedback to the developers about the quality of the system being built.
3. Some organizations already strive for continuous delivery to automate the delivery process for faster and safer delivery to customers with less effort.
4. *Clean Code:* Continuously paying attention to writing good code from the very beginning is considered more and more important. Some of the organizations started applying the Clean Code [23] approach with the goal of establishing a constantly high code quality. These teams also apply further practices like continuous quality control, regular or even institutionalized code reviews, and regular pair programming.

Management Practices

All teams in our study originally started with Scrum [17]. However, most teams have carefully altered Scrum over time and customized it to the specific needs of their organizations and projects. The ability to alter the process is important for the team for several reasons: First of all it gives the team members the feeling that they are the

masters of the process, not vice versa. It objectively allows the elimination of impediments that hinders the team of being efficient. Secondly, it shows that Scrum seems to be a good choice to start with, but that it is not sufficient for all situations and thus should be adapted to the various needs [19]. Retrospective is a good example of how to handle the *dampening* of the parts that fail and the *amplifying* of the parts that succeed.

The interviewed companies especially emphasized the following management aspects:

1. *Customer and Requirements:* In all the interviews, it was pointed out that an intensive and frequent communication with the customer was of outmost importance.
   Agile teams are implicitly or explicitly aware of the fact, that technology sometimes makes solutions possible, which previously nobody (e.g. the users) had seen. Because of this gathering user requirements can be a problem. The users do not know what they want if they do not even know that it exists. It is important to note, that software is developed in a co-evolutionary system with technology.
   In all the successful projects there was a very good understanding of the needed requirements by all team members.
   Communication solely with the Product owner is unsatisfying. Developers feel the need to communicate directly and get feedback from the end-users.
   User Stories [16] is the premier method to express functional and even non-functional requirements on cards.
2. *Agile Champion:* Leaders on all levels of agile organizations need to adopt a Catalyst Leadership style. These leaders thrive by inspiring others without losing the cohesion within the entire system. They know they can trust the organization and its individual members. Most importantly they know, at least by own experience, that software development takes place in several domains at the same time.
   In the interviews we found, that in all the projects there was a least one person that was championing agility and we therefore use the term *Agile Champion* to refer to the role of that person. Why is such a role needed? When an organization wants to become agile, change is inevitable. Of course, it would be nice if positive change happened magically with no effort. Unfortunately, experience shows that it does not. On the contrary, change is very challenging.
   What is the role of the agile champion? Table 4 lists several important activities that require huge amount of talking to people. The role of the agile champion is not to tell them what to do, but to inspire them with a vision of where they and the project could be. It is more a role of pulling than of pushing.
3. *Collaboration and Communication:* Intensive and open communication among all stakeholders is seen to be one of the key elements for successful agile projects. In our interviews we identified three major communication scenarios. The team members themselves must communicate among each other intensively. The team as a whole must communicate with the customer and end-users, and finally the team must establish good communication with the management (which is often organized in a classical hierarchical way).
   To foster communication in teams, almost all organizations implement co-located teams in an open office environment. Most communication is then carried out informally over the desk. This helps to reduce official meeting time significantly and make the necessary meetings much more efficient.
   Regular communication with the customers is established through short iterations with reviews and continuous feedback loops. However, developers have a clear need to communicate directly with end-users instead of the product owners.
   Communication and collaboration with remote teams is always seen as challenging. Companies invest a lot to establish a good communication between these teams. Means are regular chat sessions, remote participation in meetings, and video conferencing. Despite its high costs, some companies value bringing the people together physically at regular intervals. A week of common work usually "refreshes" relationships for about 5-6 weeks, and then the next physical meeting is due.

leading and inspiring agility
helping define which change is necessary
convincing and bringing on board others to support the change
showing that changes are happening and giving good results
avoiding "cowboy" agile and backsliding to the former approach (e.g. waterfall)
leading modifications to the change
removing impediments from the change
leading people to the next level if it starts to plateau

Table 4: The role of an agile champion

Agile Values

All interviewed organizations realized that developing agile is not only a matter of changing processes, but foremost, a change of the culture in an organization. Thus values become more important. Scrum defines the following five Scrum values [17] (in no particular order): *Commitment, Focus, Openness, Respect* and *Courage*. Extreme Programming [8] defines the values a bit differently: *Simplicity, Communication, Feedback, Re-*

spect and *Courage*. In the Agile Manifesto [12] the four core values are: *Individuals and their interactions, delivering working software, customer collaboration* and *responding to change.* Of course there are also other important values like trust, safety, security, quality-of-life etc.

All of these values have in common that they are important agents in complex adaptive systems, i.e. agile software development projects. Because we each interpret the values differently as individuals and as teams, we need to take a look at each value and decide as a team what that value means to us. What is most important is that the team behavior is aligned to the team values. Below we present an overview of the most important findings.

Transparency and Openess

Transparency and openness are highly valued in agile software development. Different measures are taken by the team in order to keep the organization as well as the customer informed about progress and to get feedback quickly. For instance, in Scrum there is the Scrum board and the daily standup meeting, where "the whole world" is invited to attend.

In the study we found that in all projects both aspects were very important. However, transparency can be either good or bad. Sometimes, transparency can prevent people from innovating [9]. Agile leaders know that for news things or innovative ideas to be created, sometimes privacy and protection is needed, because otherwise people outside the team look at them and there is a risk that these ideas get prematurely killed. Therefore, they create protected spaces to allow for new ideas to evolve. So, the right amount of transparency depends on the context of the project, the culture of the company and should be carefully balanced by the agile champion.

Openness of the individuals also means, that the team members not only take responsibility for a dedicated area of the project, e.g. requirements, components, etc., but are open and willed to understand the system as a whole, to have the big picture in mind. People in agile teams take responsibility for what they do in the context of the project as a whole. Agile organizations appreciate that they are (complex) systems made up of self-aware individuals and their members understand, at least implicitly, that each of their interactions may affect this system in potentially unpredictable manners. Again, the theory behind this is well described in complexity theory, i.e. complex adaptive systems. Software development is a co-evolutionary service and not a simple manufacturing process.

Another aspect of openness is the will and capability to learn. As the market place is always changing, agile organizations deliver value through the process of learning. Any change in the organization is based upon continuous learning through successful and failing experiments. In other words change happens as a sequence of learning events that combine to create paramount value, rather than by executing a master plan toward a static goal.

Organizational Culture

It is important how the organization and the company that encompass the agile team and project are organized. Principally there are three possibilities:

1. Agile team, organization and company;
2. Agile team and organization, non-agile company;
3. Agile team, non-agile organization and company.

In the study, we found that seven projects started out with the third option and after some time and effort managed to move to stage 2. In the other project both organization and company were agile. In non-agile companies we observed some tension between the different cultures. Some of the typical problems are differences in hierarchy, responsibility, openness, trust, reporting, management, compensation, planning etc. For instance, if agile values such as transparency and openness lead to promotions and praise in the company, those behaviors will be the ones which individuals select. But if they are at odds with the organizational culture, they will not, and no agile culture can evolve.

Agile project leaders know that they cannot change the company, at least not in the short term. We found that usually they respond by creating an agile environment within the company as well as possible and proactively manage the boundaries. This requires a lot of skills and patience from the agile leader.

Craftsmanship

The idea of highlighting the importance of software development practice has been popularized in [13]. The author introduces the disciplines, techniques, tools, and practices of software craftsmanship. Craftsmanship is much more than a technique: It is about attitude. The author shows how to approach software development with honor, self-respect, and pride; work well and work clean; communicate and estimate faithfully; face difficult decisions with clarity and honesty; and understand that deep knowledge comes with a responsibility to act [14].

In our interviews, we found that members of agile teams and organizations take pride in their work. They seek mastery in their (programming) skills. They know the importance of producing high quality software and great value. In the interviews, all the teams and organizations pointed out that craftsmanship, or whatever they named it, was important and that most developers had

a high motivation to become better experts. We found that they challenge themselves and they challenge their colleagues to continuously grow and develop.

Conclusion

In this paper, we have argued that software development is a multi-domain problem, i.e. problems are in the ordered or complex domain. When different people or groups of people are involved, we are typically dealing with a complex (adaptive) system. In such systems, there is no or only weak causality. This means there are no recipes for success. What worked in one project might not work in the next one, or even worse, it might be outright dangerous. When people are not aware of this, they tend to apply best practices or call in an expert. Experts and best practices only work in the ordered domain but not in the complex domain. In the complex domain, a different approach is required.

In the qualitative interviews we saw that successful agile teams and team leaders are actually applying different strategies. They are - implicitly or (seldom) explicitly - aware that software development is a multi-domain problem and act accordingly. In practice, they find that the agile method they are following does not always work and they try to find a different solution strategy. They amplify what works and dampen what does not work.

Agile teams think differently about different problems. There is no one size fits all approach and the action they take depends on which domain the problem is in.

Further work

Our interview study has generated a deep insight about successful agile projects - but also raised new questions.

Does it help the agile team to know more about complexity theory and especially its application in software development? There is the hypothesis that development teams who understand the theory of decision-making frameworks like Cynefin can in practice deliver better software products faster and cheaper. This combination of theory and practice is often referred to as *praxis*. Further research to validate or reject this hypothesis is necessary.

Organizational culture for agile teams, organizations and companies is another huge topic. There are some studies [21], [22] but further work is required. What is the influence of agile organizations in non-agile companies on key aspects like innovation, talent, motivation or customer satisfaction? What happens to innovation? How can agile and non-agile teams and organizations and companies co-exist, does the whole company have to be organized in an agile manner?

References

[1] Version One. State of Agile Development Survey results. http://www.versionone.com/state_of_agile_development_survey/11/, 14.11.2013

[2] S. Kaltenecker et. al. Successful Leadership in the Agile World – a Study Report of PAM. 2011. (German only)

[3] Martin Kropp, Andreas Meier, Swiss Agile Study - Einsatz und Nutzen von Agilen Methoden in der Schweiz. (German) www.swissagilestudy.ch, 20.10.2013

[4] David J. Snowden, Mary E. Boone A Leader's Framework for Decision Making. Harvard Business Review, November 2007, pp. 69–76

[5] D. Snowden, C.F. Kurtz. The new dynamics of strategy: Sense-making in a complex and complicated world, IBM Systems Journal, Volume 42, Number 3, 2003, pp.462-483

[6] R. Arell, J. Coldewey, I. Gat, J. Hesselberg, Characteristics of Agile Organizations, Agile Alliance, 2012, http://www.agilealliance.org

[7] M. Kropp, A. Meier, Teaching Agile Software Development at University Level: Values, Management, and Craftsmanship, CSEE&T 2013, San Francisco

[8] Kent Beck, Extreme Programming Explained: Embrace Change. Addison-Wesley, 2004 ISBN 0-321-27865-8

[9] D. Snowden, Keynote: Making Sense of Complexity, 5. Lean Agile Scrum Conference, LAS 2013, Zurich. http://www.lean-agile-scrum.ch/wp-content/files/2013/Kynote_Making%20Sense%20of%20Complexity%20-%20Dave%20Snowden.pdf

[10] Complex Adaptive Systems http://en.wikipedia.org/wiki/Complex_adaptive_system, Retrieved 18.Aug.2014

[11] http://commons.wikimedia.org/wiki/File:Cynefin_as_of_1st_June_2014.png, Retrieved 18.Aug. 2014

[12] Agile Manifesto. http://agilemanifesto.org/, Retrieved 20.Aug.2014

[13] Robert C. Martin, Clean Code: A Handbook of Agile Software Craftsmanship, 2009, ISBN 0-13-235088-2

[14] Robert C. Martin, The Clean Coder: A Code of Conduct for Professional Programmers, Prentice Hall, 2011, ISBN 0-13-708107-3

[15] Mike Cohn, Agile Estimating and Planning, 2006, ISBN 0-13-147941-5

[16] Mike Cohn, User Stories Applied, For Agile Software Development, 2004, ISBN 0-321-20568-5

[17] Ken Schwaber, Mike Beedle. Agile Software Development with Scrum, 2001, ISBN 0-13-207489-3

[18] Kent Beck, Test-Driven Development: By Example. Addison-Wesley, 2003, ISBN 0-321-14653-0

[19] Henrik Kniberg, Scrum and XP from the Trenches. How we do Scrum. An agile war story, 2007, ISBN: 978-1-4303-2264-1

[20] Alistair Cockburn, Jim Highsmith, Agile Software Development: The People Factor, Computer, vol. 34, no. 11, pp. 131-133, November, 2001

[21] Cristiano Tolfo1, Raul Sidnei Wazlawick, Marcelo Gitirana Gomes Ferreira1, Fernando Antonio Forcellini1, Agile methods and organizational culture: reflections about cultural levels, Journal of Software Maintenance and Evolution: Research and Practice Volume 23, Issue 6, pages 423–441, October 2011

[22] Juhani Iivari, Netta Iivari, The relationship between organizational culture and the deployment of agile methods, Information and Software Technology Volume 53, Issue 5, May 2011, Pages 509–520

[23] Robert C. Martin, Clean Code: A Handbook of Agile Software Craftsmanship, Prentice Hall, 2008

Agile Software-Entwicklung zum Anfassen

Agile Software-Entwicklung ist in der Zwischenzeit zum Vorgehen des Mainstreams geworden. Neben fachlicher Exzellenz ist eine offene, transparente, enge und intensive Kommunikation und Kollaboration unter allen beteiligten Stakeholdern essentielle Voraussetzung für erfolgreiche agile Projekte. Um diese enge Kollaboration effizient und effektiv durchzuführen, insbesondere bei verteilten Teams, fehlen in der Praxis jedoch noch die geeigneten Tools. Fehlende Durchgängigkeit, mangelhafte Integration und Medienbrüche führen zu grossem Mehraufwand und hemmen die Produktivität von agilen Teams. Im Forschungsprojekt „Agile Tools for Agile Methods" haben wir auf der Basis von neuester Multi-Touch-Technologie für extragrosse Bildschirme eine auf die Förderung der Team-Kollaboration optimierte „Agile Collaboration Wall"-Plattform entwickelt.

Stephanie Greiwe, Martin Kropp, Magdalena Mateescu | martin.kropp@fhnw.ch

Bereits mehr als die Hälfte der IT-Unternehmen in der Schweiz praktiziert agile Software-Entwicklung [1]. Das sagt die „SwissAgileStudy 2012", die von der *Fachhochschule Nordwestschweiz* (FHNW) zusammen mit der *Zürcher Hochschule für Angewandte Wissenschaften* (ZHAW) vorgenommen wurde [3]. Die Studie zeigt auch, dass obwohl die agile Entwicklung sich als effizienter und erfolgversprechender als herkömmliche Methoden herausstellte, den Entwicklungsteams geeignete Hilfsmittel fehlen, um den hohen Ansprüchen der agilen Vorgehensweise an Transparenz und Partizipation gerecht zu werden. Im Rahmen des Forschungsprojektes „Agile Tools for Agile Methods"[1] hat ein interdisziplinäres Forscherteam dreier Instituten der FHNW (Institut für Mobile und Verteilte Systeme, Institut für Kooperationsforschung und Institute for Competitiveness) in Interviews mit Schweizer IT-Unternehmen die Zusammenarbeit in agilen Teams untersucht. Agile Entwickler, *Scrum Master* und *Project Owner* aus elf Unternehmen wurden über agile Methoden und Praktiken, über Kommunikation- und Kollaborationsprozesse und die dabei eingesetzten Tools befragt. Analoge Pinnboards sind immer noch sehr verbreitet, es fehlt ihnen aber die Beständigkeit und die Möglichkeit der Rückverfolgung von Informationen. Bestehende Desktop- oder Web-Applikationen beheben zwar dieses Defizit, sie sind aber wenig geeignet für die Bedürfnisse hoch-kollaborativer Teams, wie dies in agilen Projekten erforderlich ist.

Merkmale agiler Kollaboration

Erfolgreiche agile Teams zeichnen sich durch eine Kultur der Transparenz, Offenheit und gleichwertiger Partizipation aller beteiligten Stakeholder aus. In solchen Teams wird sehr offen und häufig informell kommuniziert und eng zusammenarbeitet. Dabei wird der agile Prozess zielgerichtet an die jeweiligen Projektbedürfnisse angepasst. Auch in den Interviews betonten die Teilnehmer die Wichtigkeit einer offenen und möglichst häufigen Kommunikation unter allen beteiligten Stakeholdern. Relevante Informationen sollten allen Stakeholdern uneingeschränkt zur Verfügung stehen. Dies wird typischerweise durch regelmässige Treffen der Teammitglieder inklusive dem Kunden in Meetings wie *Daily Stand-Ups*, *Sprint Planning*, *Retrospective* und *Review* erreicht. Die agilen Meetings werden zielgerichtet und konsequent *timeboxed* durchgeführt, d.h. ein Meeting wird nach festgelegter Dauer abgeschlossen und offene Punkte in abgesetzten Besprechungen unter den direkt Betroffenen bereinigt. Wichtig in agilen Projekten ist, dass jedes Teammitglied sich gegenüber dem Projekt *committed* fühlt und jeder sich aktiv einbringt. Meetings lassen sich als unschlagbares Mittel der Partizipation erkennen. In den Meetings entscheidet das Team über Vorgehen, Prioritäten und Fristen im Projekt. Gleichzeitig bieten die streng strukturierten Meetings einen Rahmen um Ressourcen und Hindernisse auszudiskutieren. Es ist genau diese Kultur der Transparenz, Offenheit und Partizipation, wofür neue unterstützende Tools, agile Kollaborationsplattformen, benötigt werden.

Die Interviews und Beobachtungen zeigten, dass eine sehr grosse Brandbreite an Tools für die Zusammenarbeit zum Einsatz kommt: von *nur* physischen Pinnwänden bis zu vollständigen digitalen Umgebungen, bei denen z.B. das *Daily-Stand-Up-Meeting* mittels Projektion durchgeführt wird, an denen eine Person das digitale Tool bedient. Am häufigsten sind Mischformen anzutreffen. Dabei werden in der Regel langfristige Artefakte, wie z.B. Benutzeranforderungen (*User-Stories*), digital gespeichert, kurzfristige Artefakte, wie Entwickleraufgaben (*Tasks*), nur auf

1 Mit freundlicher finanzieller Unterstützung der Strategischen Initiative der FHNW und der wertvollen Mitwirkung der beteiligten Unternehmen. http://www.fhnw.ch/technik/imvs/forschung/projekte/si-atam/si-atam

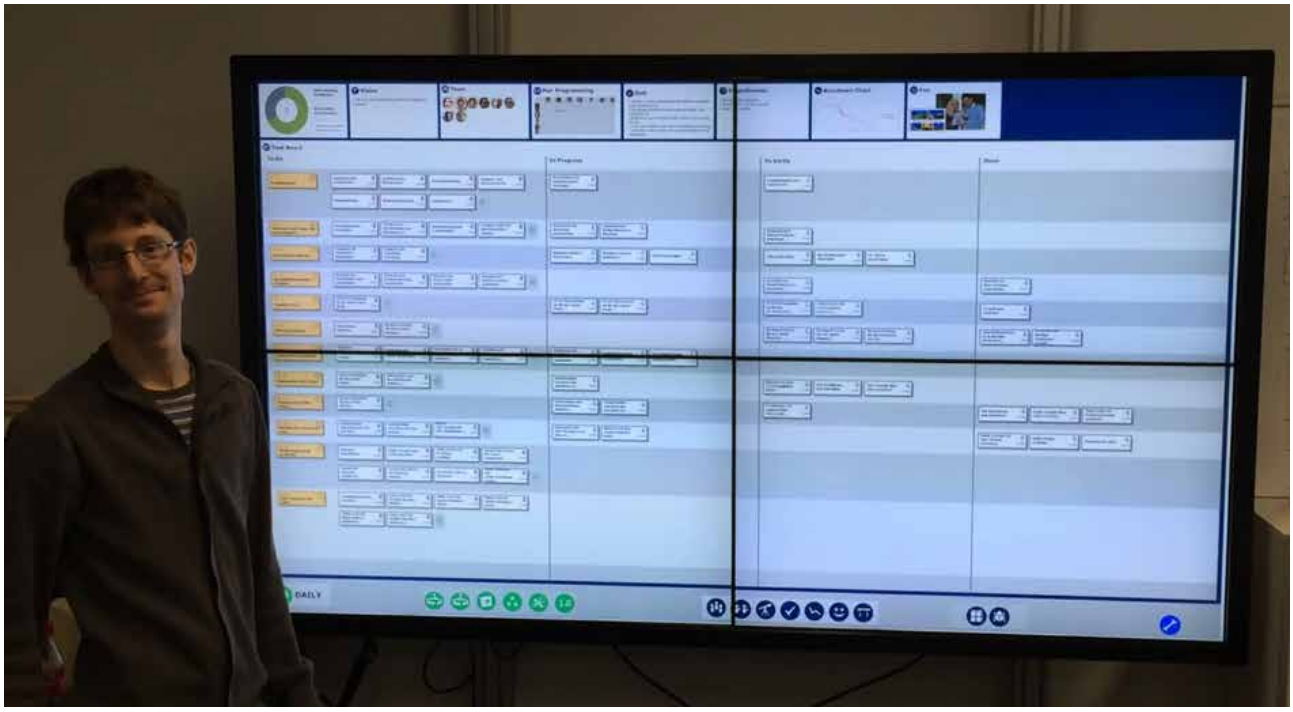

Abbildung 1: Agile Collaboration Wall

Papier erfasst. Bei physischen Pinnwänden wird vor allem die sehr grosse Flexibilität als Vorteil betont. Als weiterer positiver Effekt nannten die Interviewten die Fokussierung auf das Wesentliche, die mit dem begrenzten Platz einhergeht. Als Vorteil der digitalen Tools wird unter anderem die Ortsunabhängigkeit (Zugriff von überallher) genannt. Die Funktionalität der digitalen Tools kann in der Regel angepasst werden, Automatisierungen werden möglich. Der gleichzeitige Einsatz physischer und digitaler Tools verursacht jedoch auch Mehrkosten: Es entsteht zusätzlicher Aufwand für die häufig redundant geführte Informationen und weil die Teams mit zwei unterschiedlichen Werkzeugen arbeiten müssen (Medienbruch). Gerade bei verteilten Teams werden solche Medienbrüche problematisch. Neue Ansätze, die die Zusammenarbeit agiler Team z.B. mittels digitalen *Multi-Touch-Walls* in Kombination mit mobilen *Tablets* fördern, zeichnen sich als alternative Technologie ab. Sie können die Sichtbarkeit sowie leichte und intuitive Bedienbarkeit physischer Tools mit der Ortsunabhängigkeit und funktionalen Flexibilität von bisherigen digitalen Tools kombinieren und noch verbessern.

Aus der Analyse der Interviews lassen sich folgende essentielle Anforderungen an eine agile Kollaborationsplattform ableiten:

- Relevante Information sollten jederzeit allen Beteiligten sichtbar sein;
- Sie sollte die kurzen Kommunikationswege des agilen Arbeitens unterstützen;
- Informationen sollen kontextabhängig dargestellt werden;
- Prozessanpassungen sollen im Tool abbildbar sein;
- Visualisierungen sollen den Stakeholder-Rollen und -Bedürfnissen gerecht sein;
- Neue Funktionalität soll sehr einfach zur Plattform hinzugefügt werden können;
- Das Tool soll intuitiv zu bedienen sein;
- Das Tool soll die Zusammenarbeit nicht nur durch die Unterstützung agiler Praktiken und Meetings sondern auch durch das Ermöglichen des Parallelarbeitens fördern;
- Das Tool soll in bestehende Infrastruktur integrierbar sein;
- Das Tool soll die verteilte Entwicklung unterstützen;
- Medienbrüche müssen vermieden werden.

Wichtig dabei ist, dass die Tools die Teams in ihrem Vorgehen nicht einschränken, sondern die Zusammenarbeit im Team aktiv fördern, sehr einfach zu bedienen sind und haptische Erfahrungen für die Benutzer bieten. Die FHNW hat begonnen, die gewonnen Erkenntnisse zur Entwicklung einer kollaborativen *Multi-Touch-Agile-Wall*-Applikation umzusetzen.

Agile Collaboration Wall (aWall)

Auf der Basis der gewonnenen Erkenntnisse entwickelt das Team neue Interaktionskonzepte für extra-grosse *Multi-Touch*-Systeme, um die Kollaboration in agilen, auch verteilten, Teams zu fördern und die Vorteile von analogen Pinnwänden und der digitalen Welt zu vereinen und realisieren bereits erste Prototypen der aWall (agile Collaboration Wall) Plattform. Der Bildschirm wird zur digitalen Pinnwand, auf der das agile Team das Entwicklungsprojekt interaktiv plant und durchführt (siehe Abb. 1). Mit *Drag-and-Drop* können z.B. während *Daily-Stand-Ups*-Aufgaben verscho-

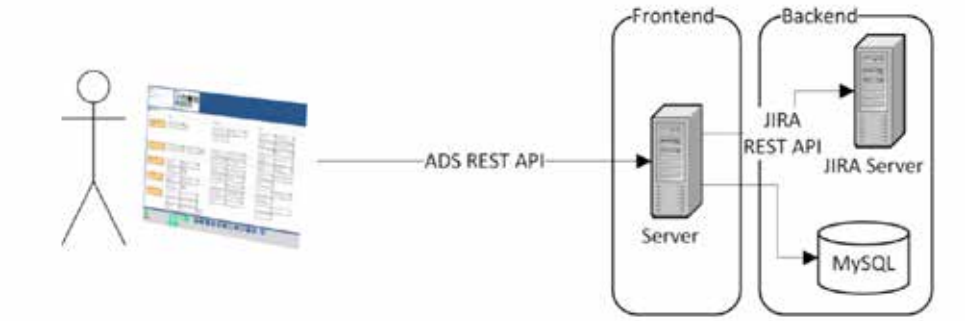

Abbildung 2: Systemarchitektur der aWall [7]

ben und Personen zugewiesen werden. Alle relevanten Informationen sind jederzeit ersichtlich und können im Team besprochen und bearbeitet werden.

Das *Multi-Touch*-System besteht aus vier 42-Zoll Bildschirmen umgeben mit einem Zwölfpunkt-Touch Infrarotrahmen der Firma PQ Labs [4]. Zur besseren Unterstützung von verteilten Teams wird die *aWall*-Plattform als Web-Applikation entwickelt. Dabei kommt als Technologie HTML5, CSS und JavaScript zum Einsatz.

Die Plattform fokussiert auf die Unterstützung der Zusammenarbeit von agilen Teams und setzt auf bestehenden *Issue-Tracker*-Systemen wie Atlassian-Jira [5] oder Microsoft TFS [6] auf. Somit können die Benutzer für anderweitige Aufgaben ihre gewohnten Tools nutzen, für die effiziente agile Zusammenarbeit jedoch die *aWall* einsetzen. Abbildung 2 zeigt die Systemarchitektur der *aWall*-Plattform, mit der sich Drittsysteme einfach integrieren lassen.

Das *Web-Frontend* greift über die *AgileData-Service* REST Schnittstelle auf die Daten der Serverseite zu. Diese Schnittstelle bildet eine Entkopplungsschicht zur Einbindung von *Issue-Trakker*-Systemen von Drittanbietern.

GUI- und Interaktionskonzept

Die Oberfläche des Boards lehnt sich stark an die Gestaltung von physischen Tools an, insbesondere des *Taskboard*, ergänzt um Elemente, welche die entsprechende Informationstechnologie für eine effiziente Arbeit ermöglicht und die Zusammenarbeit unterstützen und fördern soll. Basierend auf den erhobenen Anforderungen wird im Konzept zwischen aktiven und informativen *Views* unterschieden:

Activity Views stellen Ansichten dar, die hohe Interaktion ermöglichen. Sie werden typischerweise in Meetings verwendet, in denen das ganze Team neue Informationen hinzufügt oder bestehende modifiziert. Bei diesen Ansichten spielt daher die einfache Interaktionsmöglichkeit aber auch die übersichtlich Darstellung eine zentrale Rolle, damit die zu bearbeitende Information schnell gefunden wird.

Informative Views (Static-Views) stellen kontextabhängige ergänzende Informationen dar, die für die optimale Durchführung des jeweiligen Meetings nötig sind. Dies können zum Beispiel sein: *Vision*, *Definition-of-Done*-Liste, oder das *Burndownchart*. Solche Informationen werden von herkömmlichen Tools meist vernachlässigt.

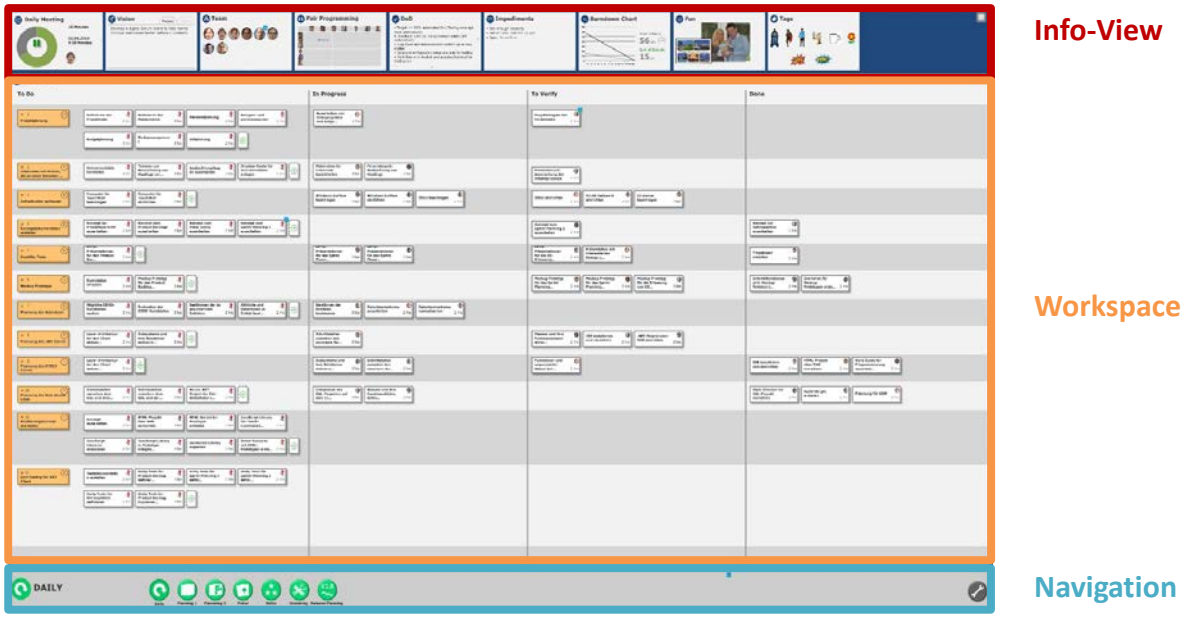

Abbildung 3: GUI Grundkonzepte

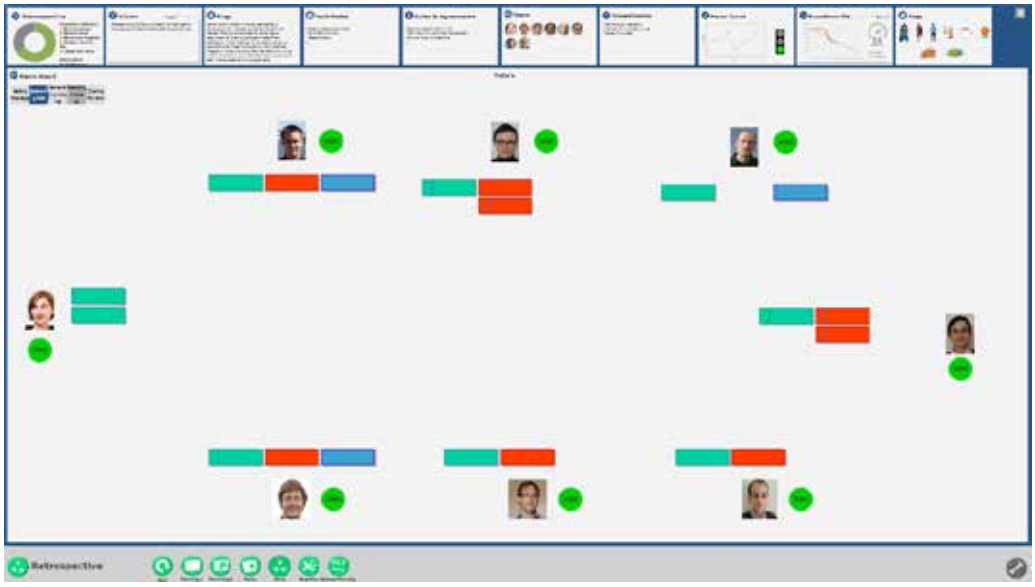

Abbildung 4: Retrospective Meeting

Jedoch spielen sie eine wichtige Rolle für das Team, da solche Ansichten das transaktionelle externalisierte Gedächtnis des Teams darstellen und wichtig für den Zusammenhang des Teams sein können. Durch deren Design und Anbindung an aktive *Views* entsprechen sie den wichtigen Anforderungen: Transparenz der Informationen, Anpassungsfähigkeit und Flexibilität des Interfaces. So können je nach Art des Meetings verschiedene passende Ansichten vordefiniert und angezeigt werden. Zusätzlich können die verschiedenen informativen Ansichten flexibel und schnell während der Zusammenarbeit entfernt oder hinzugefügt werden.

Informative Ansichten werden dabei am oberen Bildschirmrand in der *Info-View* dargestellt; *Activity Views* im zentralen Bereich des *Workspace*; hinzu kommt die Navigation am unteren Bildschirmrand (siehe Abb. 3).

Szenarien der Interaktion mit den digitalen Elementen sehen die Möglichkeit einer gleichzeitigen Interaktion für alle beteiligen Akteure vor. So sind nicht nur direkte Eingaben auf dem grossen Bildschirm möglich, sondern auch Eingaben mittels mobilen Geräten (Smartphones und Tablets). Dadurch kann sich, ähnlich wie bei physischen Werkzeugen, jeder Teilnehmer aktiv in das Meeting einbringen. Während eines *Sprint-Planning-Meetings* könnten die Teilnehmer z.B. die Tasks sowohl direkt an der interaktiven Wand als auch mittels Tablets erstellen und überarbeiten. Der kollaborative Prozess ist am Spezifikum und an den Anforderungen der jeweiligen Meetings angepasst. Während in *Dailys* die Sichtbarkeit aller Interaktionen wichtig ist, profitiert das *Retrospective-Meeting* (siehe Abb. 4) von der Privatheit der Abstimmung (so wird z.B. die Gefahr, dass ein Teammitglied sich nicht trauen würde, Kritik abzugeben dadurch gemindert).

Zusammenfassung und Ausblick

Obwohl sich agile Entwicklung als Vorgehensweise in IT Projekten etabliert hat, fehlt es vor allem an Werkzeugen, die die hoch-interaktive Zusammenarbeit in agilen Projekten geeignet unterstützen. In unserem interdisziplinären Forschungsprojekt „Agile Tools for Agile Methods" haben wir durch Firmen-Interviews untersucht, durch was sich eine erfolgreiche Kollaboration in agilen Teams auszeichnet und die Bedürfnisse an geeigneten digitalen Tools ermittelt. Diese haben wir in der Entwicklung einer *Agile-Collaboration-Wall*-Applikation für extra-grosse *Multi-Touch*-Systeme umgesetzt. Die entwickelten Prototypen und Konzepte sollen im Rahmen von weiteren Forschungsprojekten zu einer eigentlichen Kollaborationsplattform weiter entwickelt werden, die den gesamten agilen Prozess und dessen Aktivitäten unterstützt. Dazu werden auch weitere Kooperationen mit Industriepartnern angestrebt. Das Team wird seine Forschungsaktivitäten im neuen *Agile Collaboration Lab* bündeln und verstärken.

Referenzen

[1] Agile Manifesto. http://agilemanifesto.org/, Retrieved 20.10.2014.

[2] Version One. State of Agile Development Survey results. http://www.versionone.com/state_of_agile_development_survey/11/, Retrieved 14.10.2014

[3] M. Kropp, A. Meier. Swiss Agile Study – Schweizweite Studie zum Einsatz von Agilen Software Methoden in der Schweiz. http://www.swissagilestudy.ch. 2012. Retrieved 20.11.2014

[4] PQ Labs. http://multitouch.com/product_wall.html, Retrieved 1.12.2014

[5] Atlassian Jira. https://www.atlassian.com/software/jira, Retrieved 2.12.2014

[6] Microsoft TFS. http://www.visualstudio.com/en-us/products/tfs-overview-vs.aspx, Retrieved 2.12.2014

[7] Stefan Röthlisberger. Agile Multi-Touch Task Board. P8 Studierendenarbeit. 2014.

Secure Communication in a Distributed Load Management System

The transition from the traditional power generation using centralized power plants toward smaller, distributed facilities raises the importance of load management. To establish a balance between production and consumption, they have to be regulated in a dynamic fashion. Intelligent, networked systems are necessary to achieve that goal. A considerable part of the network communication is likely to be conducted over the public Internet, therefore secure communication is an important aspect. In this paper, we discuss the problems of existing systems that do not fulfill current and future security requirements, and we develop an approach that allows secure communication using the WebSocket protocol even when devices considered insecure are involved.

Peter Gysel, Matthias Krebs, Stefan Röthlisberger | peter.gysel@fhnw.ch

A lot of companies in Switzerland are already using *load manager* (LM) devices to optimize their power consumption. Their primary use case is to turn off non-critical appliances temporarily in order to prevent short peaks in electrical load (peak shaving). They thereby save money, because the price per kWh is based on the highest load peak within a 15-minute time slot.

Since such LM devices are widely deployed, they can be interesting to energy distributors. As the power grid has been constructed to support the highest peak power consumption possible, a system smoothening those peaks could potentially save a lot of money. It could also be used to help building autonomous communities. In which power is produced in small distributed power plants and consumed locally.

The Smart Grid, a modernized electricity grid that uses information and communications technology to gather and act on information, is an ongoing topic and is approached cautiously because of the high risk involved interconnecting the electricity grid for monitoring and control. It is more of an evolution than a revolution and thus the integration of legacy devices has to be considered. We are going to show how we integrate a LM device into a load management system to help an energy distributor stabilizing its power grid.

There are different scenarios where load management is used, as Figure 1 shows. Each scenario imposes different constraints on how the load can be managed. Some depend heavily on the time of the day, others just need to run for a certain amount of time per day. Different devices of several generations are being used and lots of them can be considered legacy. Nevertheless, they still work and their configuration is fine-tuned to the needs of the customer.

Legacy devices with network functionality (e.g. metering devices) often use aged communication protocols such as Modbus [1]. These protocols have been built for efficiency and robustness, but they do not offer any kind of security and thus are unsuited for communication over the Internet. Updating the devices is no option either, as they do not have the computational resources needed to implement any kind of encryption. They were designed in a time when the Internet, as we know it today, did not exist. Up to now, the problem has often been approached by tunneling all the communication through a *virtual private network* (VPN). A VPN interconnects networks which intro-

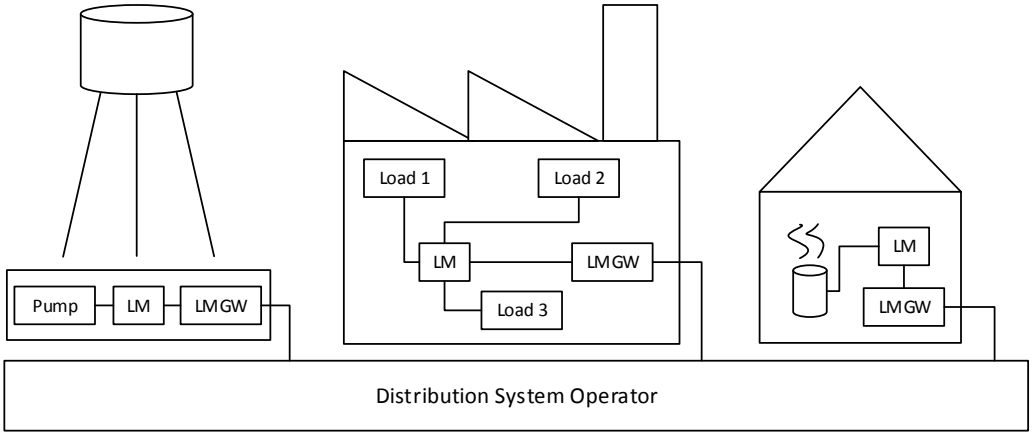

Figure 1: Overview of how our gateway can be used with different load management devices

duces additional security challenges and will be discussed later in this paper. As an example, such a solution is offered by ADS-Tec [2].

A new approach is the use of a custom secure communication channel designed specifically for load management systems, with regard to the requirements of today's IT security. The *load manager gateway* (LMGW) we develop is an embedded device that incorporates this concept. It is installed in company networks containing legacy LMs and connects to them. Unlike the LM, the LMGW is allowed to connect to the Internet.

Our scenario is as follows: A central management server, located at the *distribution system operator* (DSO), issues commands over an encrypted and authenticated channel to the connected LMGWs, which relay them to the LMs (Fig. 1). The response from the LM device is then sent back to the DSO over the same channel.

This paper focuses on the challenges of deploying a LMGW device in a corporate environment and the prototype implementation of the communication between the LMGW and DSO.

Related Work

The German Federal Office of Information Security has worked out a protection profile for a smart meter gateway [3]. The document describes the security objectives and the requirements for that. The gateway connects to the Internet and is connected to one or more smart metering devices. The goals of the gateway are protecting the privacy of the consumers, ensuring a reliable billing process and protecting the Smart Grid as a whole. As such it collects, processes, and transmits data from the connected meters. An important requirement is the usage of a security module (e.g. a smart card) [4] that provides various functions related to encryption and authentication. Another important premise required by the protection profile is that all devices communicating with the gateway have to use encryption and mutual authentication. This does not allow for legacy devices without such functionality to be incorporated into the Smart Grid. The scenario of controllable systems (in terms of load management) is mentioned but not discussed further.

The design of a secure access gateway for home area networks is considered in [5]. Their article focuses on secure, real-time remote monitoring and control of managed devices using a smartphone. The proposed system architecture also enables the managed devices to send alarms to the smartphone. The emphasis is on physical layer security of wireless networks (e.g. OFDM and GSM) and capacity challenges therein.

A system which controls connected loads based on prices is introduced in [6]. The authors use a laptop to monitor energy prices and use this information as a basis for a small localized demand response system.

In [7] the authors provide a threat analysis for advanced metering networks and formulate requirements based on those threats. In a prototype implementation they use a Trusted Platform Module (TPM) for attestation and the Xen Hypervisor [8] to allow multiple virtual machines on a host to isolate different applications from each other. Their focus lies in the isolation of different applications running simultaneously on an advanced meter to preserve their integrity and confidentiality and the attestation of the software running on it.

Lots of papers can be found about cyber security in Smart Grids which discuss security-related issues and technologies that can be used. But they do not actually specify how to apply them in practice or describe a prototype implementation.

Challenges

When thinking about implementing an LMGW, the most important aspect is the communication with the DSO (Fig. 2). In such a communication an LMGW acts as an intermediary between an LM and a DSO. A DSO sends commands to an LMGW, the LMGW then gets or sets the state of the LM. The response from the LM is sent to the DSO via the LMGW.

We use the following five requirements to drive our evaluation. They are chosen to allow for seamless adoption into a corporate environment.

1. Secure communication channel using TLS (Transport Layer Security) with end-to-end encryption and mutual authentication.
2. No obligation for any special firewall setup, especially not opening a port to allow Internet traffic into the company's network.
3. The ability to pass through intermediary proxy servers.
4. Performance, because several messages may be sent to a LMGW every second.
5. A persistent bidirectional connection (optional).

In the following paragraphs, the requirements are explained further with their respective problems to our solution domain.

To protect the data exchange between an LMGW and a DSO the connection needs to be encrypted. That means no device between an LMGW and a

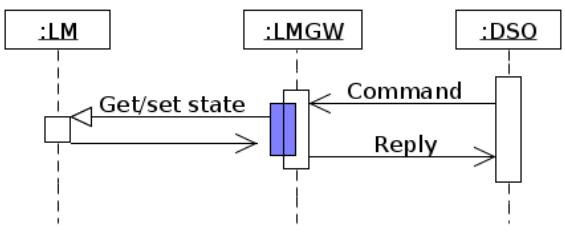

Figure 2: Communication between the DSO and a LM using the LMGW as an intermediary

DSO should be able to decrypt the traffic (end-to-end encryption).

Another important aspect of a system like a DSO is that the operator needs to be able to see who is connected. Using a proper method of authentication each LMGW has its own identity. This also allows revoking access from a specific LMGW and ensures the continuity of the system as a whole without the revoked device.

Many companies employ restrictive policies regarding network security. This means that they are not willing to lessen their security policy just for one device. Therefore, a LMGW has to adapt to this situation and has to offer a way to play nice with a restrictive network security policy. Such companies commonly use at least a firewall which by default blocks all inbound traffic from the Internet. More restrictive firewalls may even be configured to block most or all TCP/UDP ports from the company's network to the Internet, except for those used in the Web (HTTP, HTTPS). This means that a DSO cannot be the initiator of the connection.

Larger companies utilize proxies to cache, restrict and filter access to the Internet. They are located between the client and the server and may cause problems when communication protocols other than HTTP are being used, for which the proxies have not been designed.

Some companies use proxy servers featuring *deep packet inspection* (DPI), which inspect even encrypted traffic by decrypting, analyzing and re-encrypting the packets prior to forwarding. They provide a custom *certification authority* (CA) that is trusted by the clients in the company network, and instead of presenting clients the correct server certificate when they open a secure web site they present a self-signed certificate with the same server name. Because the clients trust the CA of a proxy, they accept its certificate. Communication between a client and a proxy is encrypted, but the proxy is able to decrypt the data. After analyzing the request of a client, the proxy then forwards it to the actual server, now using the correct server certificate for encryption. What the DPI system actually does qualifies as a man-in-the-middle attack. This is particularly a problem if we require end-to-end encryption for a connection.

Evaluation of Transport Protocols

When talking about the transport protocol, we mean the ISO/OSI application layer protocol used to transport the application protocol from one endpoint to another. It provides a higher level API to communicate using arbitrary messages between the two communication partners. Choosing an appropriate transport protocol for the communication between a DSO and an LMGW (Fig. 2) is crucial. We therefore analyzed some well-known approaches. We were looking for an efficient protocol which can be used without compromising the security of the company that deploys our solution. In the following sections, we discuss several protocols and their applicability to our problem domain. We compare them in terms of efficiency and how well they meet the previously discussed challenges. An overview of the discussed protocols and their fulfillment of the requirements can be found in Table 1.

HTTP Polling: HTTP is a request-response based protocol. We are looking at two approaches usually employed to communicate with a server when asking it for information. *HTTP Polling* continuously sends requests to the server in a fixed time interval. The server immediately responds either with new information or an empty response (Fig. 3a). The second approach is *HTTP Long Polling*, which is just a slight deviation from *HTTP Polling* (Fig. 3b). The difference is that *HTTP Long Polling* does not send empty responses back to the client, but instead keeps the connection open until the request can be answered with new information.

Server-Sent Events are currently being standardized as part of HTML5 by the W3C [9]. It offers a light-weight approach to push messages from the server to the client. The client initiates the connection which is basically an HTTP GET request with the *Content-Type* header set to *text/event-stream*. The server keeps the connection open and sends (pushes) multiple messages to the client until the connection is explicitly closed by the server or the client.

WebSocket is a bidirectional, full-duplex protocol using a single socket for communication (Figure 3c). It is standardized in RFC6455 [10] and is a W3C working draft [11]. A *WebSocket* client establishes a connection using the HTTP *upgrade*

Transport Protocol	efficiency	TLS	no firewall setup needed	proxy server pass-through	bidirectional
HTTP Polling	-	✓	✓	✓	-
HTTP Long Polling	-	✓	✓	✓	-
Server-Sent Events	-	✓	✓	(✓)	-
WebSocket	✓	✓	✓	(✓)	✓
Raw TCP Socket	✓	✓	(✓)	-	✓
VPN	(✓)	✓	(✓)	(✓)	✓

Table 1: Comparison of protocol capabilities

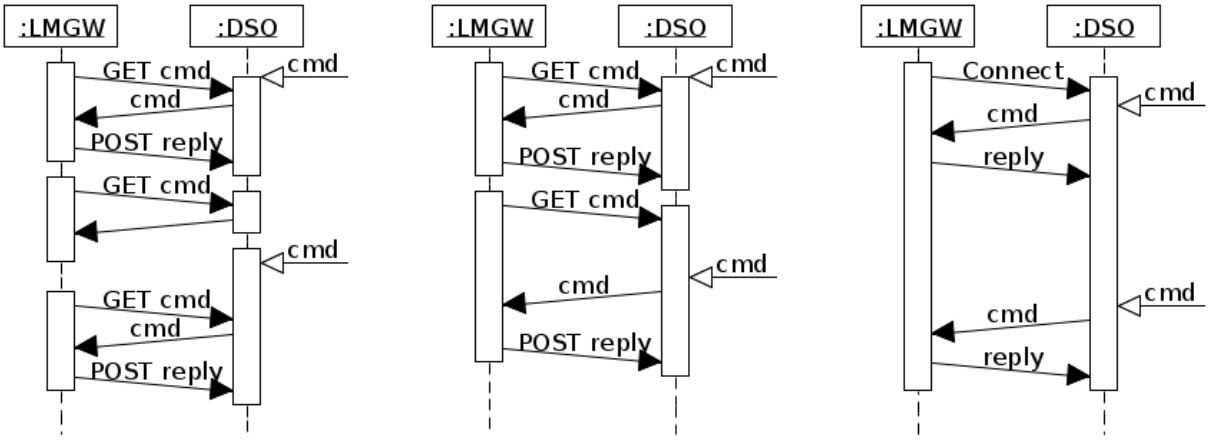

Figure 3: Communication between a distribution system operator (DSO) and a load manager gateway (LMGW). a) HTTP Polling, b) HTTP Long Polling, c) Persistent connection

header during the initial handshake. The HTTP connection is subsequently upgraded to *WebSocket*. After that, it is no longer considered an HTTP connection. When established, the connection is kept open until a participant closes it explicitly.

Raw TCP Sockets are not a transport protocol as we defined it earlier, but we have included them in our evaluation. An established TCP connection provides a bidirectional full-duplex communication channel (Fig. 3c).

Virtual Private Networks (VPN) offer an easy way to interconnect remote networks through a bidirectional tunnel. All traffic between the two endpoints is encrypted and authenticated. A client connected to a *VPN* is logically in the same LAN as the server.

Transport Layer Security

The *Transport Layer Security* (TLS) protocol ensures privacy, data integrity, and authentication using public-key cryptography. By obfuscation of the transmitted data through encryption, privacy is guaranteed. *Message Authentication Codes* (MAC) are used to detect message tampering and forgery. All of the protocols in our evaluation support TLS since it is an additional layer on top of TCP and all our protocols use TCP under the hood.

Firewall

Assuming that a firewall allows outgoing connections on all ports from the company's network to the Internet, none of the described protocols will have problems. In contrast, if the firewall is restricted to only allow Internet access over HTTP ports (TCP port 80 and 443), which is likely in many company networks, a raw TCP socket configured for a different port cannot be used. The other transport protocols use the standard HTTP ports by default, thus it is not required to open any additional ports on the firewall. If a protocol uses these ports but is in fact not based on HTTP, it might be blocked by a packet-inspecting firewall. The *WebSocket* protocol circumvents this problem by using a HTTP request to initiate the connection and subsequently uses HTTP Upgrade to switch the actual protocol.

SSL-based VPNs like OpenVPN can operate on port 443, but other technologies like IPsec require a dedicated port to be opened on a firewall and sometimes even enabling a „VPN pass-through" option (violates requirement 2).

Proxy Servers

Figure 4 shows that using *WebSocket* without encryption is likely to fail if a proxy server is involved, especially since a proxy server may close long-lived connections to avoid keeping a connection to an unresponsive HTTP server open [12]. By using TLS the intermediary proxy should not interfere with the connection.

By using TLS with an explicit proxy, the protocols using HTTP (*Server-Sent Events* and *WebSocket*) will issue an HTTP CONNECT. This method establishes a tunnel through the proxy. If the proxy allows the CONNECT, a connection can be established. Because a *raw TCP socket* and all *VPN* solutions do not issue an HTTP CONNECT, they will not get through an explicit proxy server.

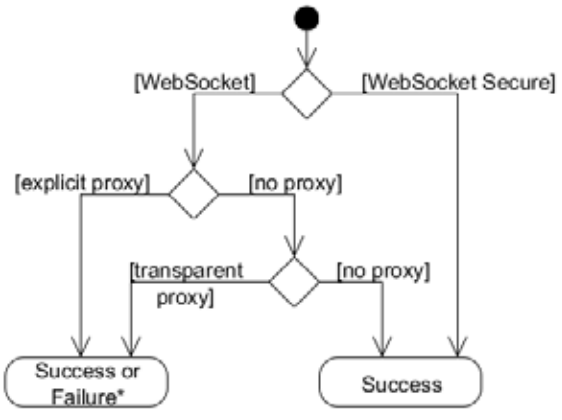

Figure 4: Shortened proxy server traversal decision tree for WebSocket [12] (*depends on the proxy's behavior and configuration)

Performance

To communicate using *HTTP Polling* in compliance with requirement 2, an LMGW would continually poll the DSO with HTTP GET requests for available messages. When a message is available on the DSO, it is included in the body of the HTTP response for the LMGW. The subsequent response from a LM is sent back to the DSO with a HTTP POST message. Another drawback of using HTTP Polling is that a message waits on the DSO to be fetched in the time between the poll requests.

When using *Server-Sent Events*, a DSO can push messages to an LMGW, because there is a persistent uni-directional connection from the DSO to the LMGW after the LMGW has initiated the connection. Nevertheless, the LMGW has to use additional HTTP POST messages to send data back to the DSO, since *Server-Sent Events* do not offer a bidirectional channel.

Using *HTTP Polling*, every POST request (i.e. polling attempt) requires establishing a new connection over TLS. A connection secured with TLS requires two additional roundtrips between the sender and receiver to exchange the certificates and negotiate the encryption. By using *Server-Sent Events* the overhead can be reduced but still requires creating a new TLS connection for every POST request. Both protocols have considerable overhead compared to a protocol offering a persistent bidirectional connection (requirement 4).

The *WebSocket* protocol and *raw TCP sockets* both offer a bidirectional, full-duplex and persistent communication channel. Therefore, both protocols use the available bandwidth responsibly, given the fact that the TLS overhead is concentrated at the time of the connection initiation.

A *VPN* connection is persistent but introduces additional traffic because it establishes a virtual network over the Internet. By using SSL-based solutions like OpenVPN all TCP/IP packets are encapsulated by the *VPN* for the transport over the Internet. The *VPN* package has a TCP and IP header and encapsulated inside is the actual package, again with TCP and IP headers.

An important aspect for the performance of a communication channel is latency. In [13] the authors compare *HTTP Polling*, *HTTP Long Polling* and *WebSocket* without TLS. The latency of polling is measured 2.3 to 4.5 times higher than with *WebSocket*. *HTTP Long Polling* has achieved both lower and higher latency in comparison to *WebSocket*, depending on the situation. Over the longest distance (Canada to Japan), the average latency of *WebSocket* is 3.8 to 4.0 times lower compared to HTTP Long Polling.

Another important performance indicator is throughput. In [14] the authors compare *HTTP Polling* with *WebSocket* in terms of overhead produced by the protocols used without TLS. The HTTP header they use is 871 bytes long, and a *WebSocket* message has an overhead of 2 bytes. They test 1000, 10000 and 100000 simultaneous requests per second to illustrate the difference. At 10000 requests per second *HTTP Polling* uses 66 Mbps, compared to 0.153 Mbps using *WebSocket*.

In a comparison between *HTTP Long Polling*, *WebSockets* and *raw TCP sockets*, the TCP socket was always faster and had more throughput than the others [15]. The larger the payload was, the larger the difference became.

Decision

Applying the requirements to the evaluated protocols, the *WebSocket* protocol fulfills the most of them as shown in Table 1. It is efficient and offers a bidirectional communications channel that the *HTTP*-based protocols do not. By using a *VPN*-based solution, the efficiency depends on the technology used. VPNs using IPSec are more efficient than SSL-based VPNs, but usually require that the firewall is configured to allow VPNs to pass through. SSL-based *VPN* solutions do not require the firewall to be configured, but they have a considerable overhead in bandwidth. In comparison to the *raw TCP socket*, *WebSocket* has the advantage to be able to pass through proxy servers and its higher-level API. A *WebSocket* implementation offers an API to send and receive messages. The *raw TCP socket* does not offer such high-level APIs. Besides establishing and closing the connection, even the low-level message has to be defined by the developer. *VPN*s also have another drawback which is not covered by Table 1: Linking whole networks with a bidirectional tunnel might include malicious traffic as happened in 2003 where the SQL slammer worm propagated through the *VPN* [16]. A similar situation could happen if a DSO's endpoint is taken over by a hacker, whereby he could gain access to all networks of connected LMGW operators.

Prototype Implementation

In a project funded by the Commission for Technology and Innovation (CTI)[1] we develop an LMGW, which is capable to securely communicate with a DSO developed by an industry partner. The LMGW device is based on an embedded Linux platform using an ARMv7 processor, similar to the Raspberry Pi. The communication based on our own XML-protocol is implemented in Java using the Jetty *WebSocket* library running on Java SE Embedded [17].

The *WebSocket* connection is secured with TLSv1.2 [18]. We use the mutual authentication feature of TLS, which is optional in the specification. This allows us to give each LMGW its own certificate that is used for authentication and authorization. The certificates are signed by a cus-

1 CTI Project "Virtuelles Kraftwerk Schweiz" - 15046.1 PFES-ES

tom Certificate Authority (CA) created solely for use in our system. Trusting our custom CA only, we can ensure that man-in-the-middle attacks are not possible because any other certificate presented by an attacker is rejected. All the certificates of the CA can be revoked. This gives us the possibility of denying access to the DSO by stolen devices. For the revocation mechanism we use a *Certificate Revocation List* (CRL) which is served to the LMGW devices and the DSO through a web server. The CRL contains a list with all revoked certificates of the CA. This mechanism adds a potential point of failure to the system because the LMGWs and the DSO need to access the CRL in order to verify the other endpoints certificate to establish the secure connection. The server that serves the CRL must be accessible over the Internet and is thus vulnerable to denial-of-service attacks.

In a period of more than six months we test the connection between one DSO and several LMGWs. The connection between the LMGWs and the DSO is stable and one LMGW connection uses only a combined mean bandwidth of 43.39 kbps measured for 70 hours during live operation. The DSO sends four messages per second to each LMGW device in average. The mean upload from one LMGW to the DSO is 25.66 kbps and the corresponding mean download is 17.73 kbps. The reason for the discrepancy between the upload and download bandwidth is the response from an LMGW contains more data than the request sent by a DSO.

The unsecured connection between the LMGW and legacy LM devices is a problem our solution cannot provide a solution for. That risk can be mitigated by the company itself by not allowing any connections form the Internet to an LM through the corporate firewall or even separate LMs into their own VLANs.

Conclusions and Future Work

Our concept of using the *WebSocket* protocol in conjunction with encryption is a promising approach for both secure and efficient communication over the public Internet. Using Transport Layer Security (TLS) for end-to-end encryption in combination with revocable certificates for authentication, communication is secure against man-in-the-middle attacks, e.g. message tampering, data manipulation and eavesdropping.

There are only few remaining risks, such as the vulnerability to denial-of-service (DoS) attacks due to CRL checking during certificate validation. We plan to address these issues in future development, for example through the introduction of the *Online Certificate Status Protocol* (OCSP) to validate certificates. It uses a so called OSCP responder that knows the condition of all certificates in a CA. Communication usually takes place over HTTP, implementing the request-response paradigm. If an application wants to verify a certificate it asks an OCSP responder whether the certificate has been revoked. This reduces the overhead compared to the CRL approach, because the client only asks whether the presented certificate is valid.

References

[1] The Modbus organization. The Modbus standard specification. http://www.modbus.org

[2] ADS-Tec Big-Linx Remote Service Cloud using OpenVPN. http://www.ads-tec.de/industrial-it/cloud-big-linx/big-linx.html

[3] Federal Office for Information Security, Germany. Protection Profile for the Gateway of a Smart Metering System (Smart Meter Gateway PP). https://www.bsi.bund.de/DE/Themen/SmartMeter/Schutzprofil_Gateway/schutzprofil_smart_meter_gateway_node.html, Version 1.2, 2013.

[4] Federal Office for Information Security, Germany. Protection Profile for the Security Module of a Smart Meter Gateway (Security Module PP). https://www.bsi.bund.de/DE/Themen/SmartMeter/Schutzprofil_Security/security_module_node.html, Version 1.0, 18 March 2013.

[5] Tongtong Li, Jian Ren, and Xiaochen Tang, Michigan State University. Secure Wireless Monitoring and Control Systems for Smart Grid and Smart Home. IEEE Wireless Communications, June 2012.

[6] Michael LeMay, Rajesh Nelli, George Gross, and Carl A. Gunter, University of Illinois Urbana-Champaign. An Integrated Architecture for Demand Response Communications and Control. Proceedings of the 41st Hawaii International Conference on System Sciences, 2008.

[7] Michael LeMay, George Gross, Carl A. Gunter, Sanjam Garg, University of Illinois Urbana-Champaign. Unified Architecture for Large-Scale Attested Metering. Proceedings of the 40th Hawaii International Conference on System Sciences, 2007.

[8] Xen Project, http://www.xenproject.org/

[9] Ian Hickson. Server-Sent Events, W3C Candidate Recommendation. http://www.w3.org/TR/eventsource/, 2012.

[10] I. Fette, A. Melnikov. The WebSocket Protocol. http://tools.ietf.org/html/rfc6455, Dezember 2011.

[11] Ian Hickson. The WebSocket API, W3C Candidate Recommendation. http://www.w3.org/TR/websockets/, 20 September 2012.

[12] Peter Lubbers. How HTML5 Web Sockets Interact With Proxy Servers. http://www.infoq.com/articles/Web-Sockets-Proxy-Servers, 16 March 2010.

[13] Victoria Pimentel, Bradford G. Nickerson. Communicating and Displaying Real-Time Data with WebSocket. IEEE INTERNET COMPUTING, July/August 2012, pp. 45-53.

[14] Peter Lubbers, Frank Greco. HTML5 Web Sockets: A Quantum Leap in Scalability for the Web. http://soa.sys-con.com/node/1315473, March 2010.

[15] Sachin Agarwal , NEC Europe Ltd., NEC Europe Laboratories. Real-time Web Application Roadblock: Performance Penalty of HTML Sockets. IEEE ICC 2012 - Communication QoS, Reliability and Modeling Symposium, 2012, pp. 1225-1229.

[16] North American Electric Reliability Council. SQL slammer worm lessons learned for consideration by the electricity sector. http://www.utexas.edu/law/journals/tlr/sources/Issue 90.1/Thompson/NAERC Slammer Report.pdf, 2003.

[17] Jetty. http://eclipse.org/jetty/

[18] T. Dierks, E. Rescorla. The Transport Layer Security (TLS) Protocol Version 1.2. http://tools.ietf.org/html/rfc5246, August 2008.

Man-in-the-Middle: Analyse des Datenverkehrs bei NFC-Zahlungen

Das Bezahlen mit kontaktlosen Kreditkarten liegt im Trend, insbesondere seit grosse Ladenketten wie Migros, Coop oder Valora diese Bezahlmöglichkeit unterstützen. Mit Google Wallet, Apple Pay und Tapit von Swisscom besteht vermehrt auch die Möglichkeit mittels Smartphone kontaktlose Zahlungen auszulösen. Wir haben die Daten, die zwischen einem Terminal und einer Karte bzw. einem Mobiltelefon ausgetauscht werden, bei einer echten Bezahlung aufgezeichnet. In diesem Artikel beschreiben wir dieses Protokoll und die Software, die nötig ist, um solche Daten aufzuzeichnen.

Christof Arnosti, Dominik Gruntz | dominik.gruntz@fhnw.ch

Immer mehr Schweizerinnen und Schweizer zahlen kontaktlos. Bei diesen Transaktionen wird die Karte an das Terminal gehalten. Bereits nach weniger als einer Sekunde ist der Vorgang abgeschlossen, da bei Beträgen unter 40 Franken keine PIN eingegeben werden muss. In den folgenden Kapiteln werden wir zeigen, wie eine Transaktion zwischen kontaktloser Kreditkarte und Terminal abläuft und insbesondere welche Daten dabei ausgetauscht werden. Dazu haben wir eine kleine Android-Applikation geschrieben, die es erlaubt, die über NFC übertragenen Daten zu protokollieren. Diese Applikation kann auch verwendet werden, um mit einem Relay-Angriff unerlaubte Zahlungen durchzuführen. In einem abschliessenden Kapitel gehen wir auf die Sicherheitsrisiken ein, die durch diese modernen Zahlungsmöglichkeiten entstehen.

Europay, MasterCard und Visa

Der Standard, der hinter dem Bezahlen mit modernen Kredit- und Bankkarten steht, heisst Europay, MasterCard and Visa (EMV). Er wird von der Firma EMVCo LLC betreut, die zu gleichen Teilen den sechs Kreditkartenunternehmen American Express, Discover, JCB, Mastercard, UnionPay und Visa gehört. Im Gegensatz zu früheren Kartentypen mit Magnetstreifen wird bei EMV auf der Karte ein Chip verwendet, der Berechnungen durchführen und Daten speichern kann. Zwischen dem Kartenleser (auch *Point of Sale*, POS) und der Karte (auch *Integrated Circuit Card*, ICC genannt) wird eine serielle Verbindung aufgebaut, über die der Leser Anfragen sendet, die von der Karte beantwortet werden. Diese in der EMV-Spezifikation definierte serielle Verbindung kann entweder über eine Kontaktfläche oder per NFC aufgebaut werden.

Implementiert wird der EMV-Standard von verschiedenen Firmen unter verschiedenen Namen. Im europäischen Markt sind auf der Karten-Seite *Visa Smart Debit Card* (VSDC) und *M/Chip* von Mastercard wichtig. Für die verschiedenen Verbindungstypen werden wiederum Marktnamen vergeben: *Chip&PIN* ist die gängige Bezeichnung für EMV-Transaktionen, deren Verwendungskonzept auf dem Einführen der Karte in einen Kartenleser und das Bestätigen der Karteninhaberanwesenheit durch eine PIN-Eingabe basiert. *PayPass* und *PayWave* sind die Markennamen von MasterCard und Visa für das kontaktlose bezahlen.

Grundsätzlich ist eine Bankkarte nichts weiter als eine SmartCard. SmartCards werden auch für diverse andere Aufgaben verwendet, zum Beispiel als SIM-Karten im Mobilfunk, als Zutrittskarten für Gebäude oder zum Speichern von kryptografischen Geheimnissen. Moderne SmartCards haben ein installiertes Betriebssystem wie z.B. JavaCard. Diese ressourcenschonende Java-Version enthält eine abgespeckte Standard-Library, die vor allem kryptografische Funktionen, Funktionen für die Kommunikation über die Chip-Schnittstelle und über NFC sowie Funktionen für spezielle Persistenz- und Transaktionsoperationen anbietet. Bei der Ausführung von Applets auf der SmartCard gilt, dass das Applet bei der Installation einmalig initialisiert wird und anschliessend alle nicht speziell im flüchtigen Speicher angelegten Felder automatisch persistiert werden. Bei einem Stromunterbruch – der bei jedem Entfernen der Karte aus dem Kartenslot oder aus dem NFC-Feld stattfindet – werden die Daten gespeichert.

SmartCard-Kommunikation

Das EMV-Protokoll basiert auf der standardisierten SmartCard-Kommunikation (ISO 7816 für kontaktbehaftete und ISO 14443 für kontaktlose Kommunikation). Das serielle Protokoll arbeitet mit Application Protocol Data Units (APDUs), wobei Command-APDUs vom Leser an die Karte gesendet werden, welche die Karte mit Response-APDUs beantwortet.

CLA	INS	P1	P2	Lc	Data	Le
Header				Body (Optional)		

Data	SW1	SW2
Body (Optional)	Trailer	

Abbildung 1: Command-APDU (oben), Response-APDU (unten)

Eine Command-APDU (Abb. 1 oben) besteht aus einem Header und einem Body. Im Header sind die Befehlsklasse (CLA), die Instruktion (INS) sowie zwei Parameter (P1, P2) vorhanden, wobei jedes der vier Felder ein Byte lang ist. Der Body ist von variabler Länge, und enthält drei Felder: Die zusammengehörenden Kommandolänge (Lc, ein Byte) und Kommandodaten (Data, von der durch Lc vorgegebenen Länge) sowie die erwartete maximale Länge der Antwortdaten (Le, ein Byte 00 bedeutet maximal 256 Bytes in der Antwort). Die Kommandolänge und der Data-Abschnitt zusammen sowie die erwartete Antwortlänge sind optional, so dass sich vier verschiedene Formate ergeben, die als Case 1 bis Case 4-Commands bezeichnet werden.

Eine Response-APDU (Abb. 1 unten) besteht aus einem optionalen Body sowie einem obligatorischen Trailer. Der Trailer enthält zwei Status-Bytes (SW1, SW2). Diese zeigen auf, ob das Kommando erfolgreich abgeschlossen wurde. Im Misserfolgsfall werden Details zum Fehler in diesen Feldern codiert, unter anderem die Information ob der nichtflüchtige Speicher auf der Karte verändert wurde. Im Body können beliebige Daten übertragen werden.

EMV-Protokoll

Eine EMV-Transaktion besteht aus vielen Teilschritten. Im Folgenden beschreiben wir jene Schritte eines normalen Bezahlvorgangs, die eine Kommunikation mit der Karte beinhalten:

- Application Selection
- Initiate Application Processing
- Read Application Data
- Offline Data Authentication

In jedem dieser Schritte werden eine oder mehrere Command-APDUs vom Terminal an die Karte geschickt und von dieser mit einer Response-APDU beantwortet. Die Antwort wird im Body der Response-APDU im Format *Basic Encoding Rules Tag-Length-Value* (BER-TLV, ITU-T X.690) abgelegt. TLV ist ein Format, in dem jeder Eintrag aus einem Tag (Bei EMV ein bis zwei Byte), welches das Datenformat sowie die Bedeutung der Daten beschreibt, seiner Länge sowie den eigentlichen Daten besteht. Diese Daten können wiederum im Format TLV codiert sein, d.h. die TLV-Codierung kann rekursiv verwendet werden.

Das Protokoll wird hier anhand der Anfragen und Antworten erklärt, die bei der Verwendung einer Prepaid-MasterCard als kontaktloses Zahlungsmittel bei einem Detailhändler aufgezeichnet worden sind (Abb. 2). Für jeden Schritt werden neben einer Erklärung jeweils die Anfrage als Hex-codierte Byte-Zeichenfolge und die Antwort als TLV-Decodierte Datenstruktur angegeben. Grosse Hilfen bei der Protokollanalyse sind die beiden Webseiten EMV-Lab [ELB], auf der ein Werkzeug zum decodieren von TLV-Datenstrukturen enthalten ist, sowie TVR-Decoder [TVR], ein Tool zur Decodierung von EMV-Datentypen.

Application Selection

Im ersten Schritt der Kommunikation wird durch das Terminal zuerst das *Payment System Environment* (PSE) Applet selektiert. Wenn ein Applet selektiert wird, dann ist dieses Programm für alle weiteren Anfragen zuständig, bis entweder ein anderes Applet selektiert oder die Karte vom Strom getrennt wird. Dieser Selektionsbefehl sieht immer gleich aus (Listing 1) und enthält neben dem Selektionsbefehl (00A40400) im Body die Application-ID des PSE-Applets (2PAY.SYS.DDF01, wenn diese ID als ASCII-Zeichenkette interpretiert wird). Das PSE-Applet antwortet auf den Selektionsbefehl mit einer Liste von vorhandenen EMV-Applets. Pro Eintrag sind mindestens eine Application-ID, eine Priorität sowie ein sprechender Name enthalten. Im Beispiel ist bloss der Eintrag für ein MasterCard-Applet enthalten.

Mit einer zweiten Command-APDU wird nun das gewünschte EMV-Applet selektiert (Listing 2). Falls das PSE-Applet mehrere EMV-Applets zur Wahl anbietet, so erfolgt die Wahl des zu verwen-

Abbildung 2: Der Bezahlvorgang beim Detailhändler ist erfolgreich, als letzter Log-Eintrag wird die kryptografische Bestätigung durch die Karte angezeigt.

```
Anfrage:
00A40400 0E 325041592E5359532E4444463031 00

Antwort:
6F  File Control Information (FCI) Template
|  84  Dedicated File (DF) Name : Name des aufge-
|  |      rufenen Applet, stimmt mit der Anfrage überein
|  |  325041592E5359532E4444463031
|  A5  FCI Proprietary Template
|  |  BF0C  FCI Issuer Discretionary Data
|  |  |  61  Application Template
|  |  |  |  4F  Application Identifier (AID) – card
|  |  |  |  |  A0000000041010
|  |  |  |  87  Application Priority Indicator
|  |  |  |  |  1
|  |  |  |  50  Application Label
|  |  |  |  |  M a s t e r C a r d
```

Listing 1: Selektion des Payment System Environment Applets

denden EMV-Applets entweder automatisch über die Priorität oder durch den Benutzer am Terminal. Neben den Daten, die bereits in der ersten Antwort zurückgegeben worden sind, wird bei dieser zweiten Antwort zusätzlich noch die auf der Karte gespeicherte Sprachpriorisierung zurückgeliefert.

Initiate Application Processing

Im zweiten Schritt *Initiate Application Processing* wird die eigentliche Transaktion mit dem selektierten EMV-Applet gestartet. Das Terminal frägt dazu zuerst die Prozessoptionen der Karte ab (Listing 3). Falls die Karte im letzten Schritt in den *File Control Information* (FCI) bereits Optionen definiert hat, die vom Terminal abgefragt werden sollen, so werden diese Daten als Body der Anfrage an die Karte gesendet, andernfalls (wie in unserem Fall) wird der Platzhalter 8300 gesendet.

Die Antwort enthält das *Application Interchange Profile* (AIP) und den *Application File Locator* (AFL). In ersterem sind in einer Bitmaske codiert die Fähigkeiten der EMV-Implementierung auf der Karte enthalten; in unserem Fall, dass die Karte *Cardholder Verification* und *Terminal Risiko-Management* unterstützt, sowohl mittels Chip als auch über den Magnetstreifen verwendet werden kann, und was für Datenverifizierungsmodi implementiert sind. *Terminal Risiko-Management* bedeutet, dass das Terminal Einschränkungen bei der Transaktion machen darf, z.B. einen Maximalbetrag festlegen oder bei zufällig ausgewählten Transaktionen eine Online-Verifizierung erzwingen darf. Die *Cardholder Verification* ist der Vorgang, dass die Karte bei der Transaktion bestimmen kann, wie der Kartenbesitzer seine Anwesenheit bestätigen muss, z.B. per Unterschrift oder PIN-Eingabe. Im *Application File Locator* sind in einer EMV-spezifischen Kodierung die Speicherorte angegeben, an der die für die Transaktion benötigten Daten zu finden sind.

```
Anfrage: 00A40400 07 A0000000041010 00

Antwort:
6F  File Control Information (FCI) Template
|  84  Dedicated File (DF) Name
|  |  A0000000041010
|  A5  FCI Proprietary Template
|  |  50  Application Label
|  |  |  M a s t e r C a r d
|  |  87  Application Priority Indicator
|  |  |  1
|  |  5F2D  Language Preference
|  |  |  e n d e f r i t
```

Listing 2: Selektion des EMV Applets

Read Application Data

Im Schritt *Read Application Data* werden diejenigen Datensätze ausgelesen, die im AFL angegeben sind. Dieser datenaufwändige Schritt wird in Listing 4 nur auszugsweise dargestellt. Für das Auslesen der Datensätze werden mehrere Anfragen nach dem Schema 00B2XXYY00 durchgeführt, wobei das Kommando B2 als Read Record interpretiert wird. Die beiden Bytes XX und YY bezeichnen den zu lesenden Datensatz.

Die Antworten sind wiederum TLV-Codiert. Die erste Hierarchieebene wird mit dem Tag 70 *(EMV Proprietary Template)* markiert, darunter finden sich die einzelnen Datensätze. Zur einfacheren Lesbarkeit werden die einzelnen Antworten in Listing 4 zu einer einzigen zusammengefasst.

In den von der Karte zurückgelieferten Daten sind zwei Public Keys zu finden: Der eine gehört dem Kartenherausgeber, der andere gehört zu einem Schlüsselpaar, das auf der Karte selbst abgelegt ist. Das Zertifikat des auf der Karte selbst gespeicherten Schlüssels ist mit dem Herausgeberschlüssel signiert, um ihn zu beglaubigen. Neben diesen Daten werden jedoch auch die Kreditkartennummer und das Ablaufdatum ausgegeben, ohne dass wir irgendwelche Tricks anwenden müssen. Das heisst, diese Angaben können über die NFC Schnittstelle ohne Probleme ausgelesen werden. In der *Card Risk Management Data Object List 1* ist hinterlegt, welche Daten vom Terminal an die Karte übertragen werden müssen, damit diese die Transaktion bestätigen kann. Mit der *Cardholder Verification Method* und den *Issuer Action Codes* werden die durch die Karte erlaubten bzw. erforderten Transaktionssicherheitsmechanismen beschrieben.

Offline Data Authentication

Die Daten, die mit der *Card Risk Management Data Object List 1* durch die Karte angefordert werden, werden vom Terminal zusammengestellt

```
Anfrage: 80A80000 02 8300 00

Antwort:
77  Response Message Template Format 2
|  82  Application Interchange Profile (AIP)
|  |  1980
|  94  Application File Locator (AFL)
|  |  08010100100101011004040018010100200 10200
```

Listing 3: Initiate Application Processing

```
Anfrage: 00B2XXYY 00

Antwort:
70 EMV Proprietary Template
  5A Application Primary Account Number (PAN) : Kreditkartennummer
    XXXXXXXXXXXX6317
  5F24 Application Expiration Date : YYMMDD, Kreditkartenablaufdatum
    160430
  8C Card Risk Management Data Object List 1 (CDOL1) : TLV-Tags von Daten, die zum Ezeugen des Kryptogramms
                                                       vom Terminal an die Karte gesendet werden müssen
    9F02069F03069F1A0295055F2A029A039C019F37049F35019F45029F4C089F3403
  8E Cardholder Verification Method (CVM) List : Cardholder Verification Method, in diesem Fall ausschliesslich
                                                 Online-PIN (Überprüfung der PIN durch einen Server des Kreditkartenanbieters.
                                                 Andere Möglichkeiten wären z.B. Unterschrift oder keine Verifikation nötig)
    000000000000000042031E031F03
  9F0D Issuer Action Code – Default : beschreibt die erlaubten Operationen bei einem nicht-online-fähigen Terminal
    F450840000
  9F0F Issuer Action Code – Online : beschreibt die erlaubten Operationen bei einem online-fähigen Terminal
    F470848000
  9F32 Issuer Public Key Exponent / 92 Issuer Public Key Remainder / 90 Issuer Public Key Certificate :
                                                                      Public Key und Zertifikat des Kartenherausgebers
    3 / AA159AD2B21FE1196403B51EC1 … / 474BCDC46F0B12709F6A1DD4 …
  9F47 Integrated Circuit Card (ICC) Public Key Exponent / 9F48 Integrated Circuit Card (ICC) Public Key
  Remainder / 9F46 Integrated Circuit Card (ICC) Public Key Certificate : Public Key und Zertifikat des auf der Karte
                                                                          verwendeten Schlüssels
    3 / 6A78A538D815D79F3C05 / B24B6B1AC2F0046A6…
```

Listing 4: Read Application Data

und an die Karte übertragen. Diese Daten (Tabelle 1) beinhalten Informationen über das Terminal, kryptografische Daten zur Transaktion (Nonce) sowie Informationen zur Transaktion.

Zuerst wird durch das Terminal anhand der *Issuer Action Code List*, der *Cardholder Verification Method List*, Transaktionseigenschaften (z.B. sind bei NFC-Karten in der Schweiz Zahlungen unter CHF 40 ohne PIN-Eingabe möglich, höhere Beträge erfordern jedoch eine PIN) sowie von im Terminal gespeicherten Action Codes und den Terminal-Fähigkeiten (z.B. Onlinefähigkeit, Vorhandensein eines PIN-Eingabegerätes) entschieden, ob und wie die Anwesenheit des Kartenhalters bestätigt werden muss. Die gewählte Prüfmethode wird als *Cardholder Verification Result* codiert und muss von der Karte bestätigt werden. Die eigentliche Verifizierung der Anwesenheit findet dann erst später statt.

80AE5000	SmartCard-Befehl (proprietär), Parameter P1 (hier: 50) gibt den Typ der gewünschten Bestätigung an (Offline-Signatur, kombinierte dynamische Datenauthentifizierung und Kryptogramm-Generierung)
000000000095	Transaktionsbetrag (0.95)
000000000000	Transaktionsgebühren (0.00)
0756	Ländercode (CH)
0000000000	Resultat der Kartenüberprüfung (erfolgreich)
0756	Währungscode (CHF)
141021	Zahlungsdatum (21.10.2014)
00	Transaktionstyp (Einkauf)
4CF7ED0B	Nonce (Number Once, eine Zufallszahl für die Kryptografie)
25	Terminal-Type(kundenbedientes Offline-Terminal mit Online-Fähigkeiten)
0000	Data Authentication Code (für ältere Sicherheitsprotokolle verwendet)
0000000000000000	dynamische Nr. der Karte (immer 0)
1F0302	Cardholder Verification Result, gewählte Methode (Verzicht auf Verifizierung)

Tabelle 1: Anfrage für Offline Data Authentication

In der Antwort der Karte (Listing 5) sind im *Cryptogram Information Data* Informationen über den Typ des mitgelieferten Kryptogramm enthalten (AAC: Transaktion abgelehnt, TC: Transaktion offline bestätigt, ARQC: Online-Autorisierung benötigt). Diese bestimmen zusammen mit der Terminal-Entscheidung den weiteren Ablauf der Transaktion. In unserem Fall ist dies ein Request für die Online-Autorisierung ohne weitere Kundeninteraktion.

Ebenfalls in der Antwort enthalten sind ein Transaktionszähler, der von der Karte nachgeführt wird, und die Issuer Application Data. Letztere werden unverändert an den Kartenherausgeber weitergeleitet. Der Transaktionszähler ist wohl rein informativer Natur und scheint nicht für die Validierung der Zahlung verwendet zu werden.

Die signierten Applikationsdaten sind verschlüsselt und können vom Terminal zur Bestätigung der Daten anhand des Public Keys der Karte überprüft werden. Mit den Informationen, die

```
77 Response Message Template Format 2
  9F27 Cryptogram Information Data
    80
  9F36 Application Transaction Counter (ATC)
    011D
  9F4B Signed Dynamic Application Data :
       enthält das von der Karte erzeugte Kryptogramm
    31F640F15C3E5434978…
  9F10 Issuer Application Data
    0210A0400322300000000000000000000FF
```

Listing 5: Antwort der Offline Data Authentication

dem Terminal nun vorliegen, kann die Zahlung abgeschlossen werden. Das Terminal nimmt die *Cardholder Verification* vor und führt den Online-Teil der Transaktion aus, bei der auch die signierten Applikationsdaten an den Kartenherausgeber übertragen werden.

Im Fall von NFC-Karten ist vorgeschrieben, dass eine mögliche PIN-Überprüfung bei Beträgen über CHF 40 oder € 25 online durch den Herausgeber vorgenommen werden muss. Dies ist nötig, da bei einer Offline-Überprüfung durch die Karte während dem Eingeben der PIN weiterhin eine Verbindung zur Karte bestehen müsste oder die PIN bei einer zweiten Berührung durch die Karte überprüft werden müsste.

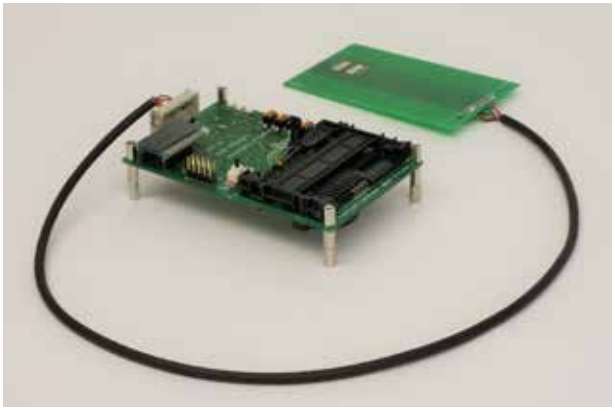

Abbildung 3: Der Smart Card Detective.
Foto von http://www.smartcarddetective.com

Man-in-the-Middle

Für das Aufzeichnen der übertragenen Daten haben wir ein Man-in-the-Middle-System verwendet. Bereits bei Kontakt-Kreditkarten kamen Angreifer auf die Idee, Man-in-the-Middle-Angriffe zwischen der Karte und dem Terminal durchzuführen. Dazu wird spezielle Hardware verwendet, die zwischen dem Kreditkarteninterface und der Karte eingesetzt wird. Darauf befindet sich typischerweise eine serielle Schnittstelle zum Auslesen und Manipulieren des Datenverkehrs oder ein Microcontroller, der dies direkt auf der Hardware erledigt. Abbildung 3 zeigt eine Open-Source-Implementierung eines solchen Man-in-the-Middle Gerätes, den Smart Card Detective, der von Omar Choudary im Rahmen seiner Master-Thesis entwickelt wurde [OSC10].

Mit dem Aufkommen der kontaktlosen Karten sowie dem Bezahlen mittels Smartphone hat sich diese Situation geändert, spezielle Hardware für die Kreditkarten-Schnittstelle ist nicht mehr nötig: Seit der Version 4.4 (KitKat) unterstützt Android Host Card Emulation (HCE), mit der eine NFC-Karte emuliert werden kann [HCE13]. HCE ermöglicht das Implementieren eines Service, der die APDUs eines NFC-Terminals entgegennimmt und beantwortet.

Die Idee hinter dem Man-in-the-Middle-System ist einfach (Abb. 4): Anstatt dass direkt die Karte an den Kartenleser gehalten wird, wird ein Android-Smartphone mit einer Proxy-Software verwendet. Der darauf implementierte Service („Terminal to Wireless“) reagiert auf die Daten, die vom NFC-Leser gesendet werden, und schickt sie über das Netzwerk an ein zweites NFC-Handy, das wiederum an eine NFC-Kreditkarte gehalten wird. Eine auf dem zweiten Handy installierte Anwendung („Wireless to Card“) empfängt diese Anfragen und sendet sie an die Karte. Die Antworten der Karte werden daraufhin über diese beiden Applikationen wieder zurück an das Terminal übermittelt. Damit ist es einerseits möglich, ohne physisch anwesende Karte zu bezahlen, andererseits ist es auch möglich, den Datenverkehr für den Bezahlvorgang aufzuzeichnen und zu manipulieren. Das wirklich Problematische daran ist, dass auf diese Art eine fremde NFC-Kreditkarte angesprochen werden kann, die sich unter Umständen weit entfernt vom Terminal befindet.

Bereits vor Android 4.4 gab es bei Android-Telefonen mit modifiziertem Betriebssystem die Möglichkeit NFC-Karten zu emulieren [FHMM11]. Während bei uns erste Versuche für Man-in-the-Middle-Angriffe mit modifiziertem Betriebssystem im Jahr 2013 fehlschlugen, gelang dies anderen Forschungsteams. Einige Studierende an der *University of Texas* hielten ihre erfolgreiche Transaktion in einem Video fest [UTC12].

Neu an unserer Software ist, dass im Gegensatz zu unseren früheren Versuchen und den Applikationen von anderen Teams standardmässig durch Android angebotene Schnittstellen verwendet werden, so dass der Angriff auf aktuellen Android-Smartphones durch die blosse Installation der entwickelten Software möglich ist. Sie bietet die Möglichkeit, zwischen dem Terminal und der Karte mitzulesen, um das in der Praxis verwendete EMV-Protokoll zu dokumentieren und zu analysieren (Listing 6). Durch einfache Anpassungen an der entwickelten Software könnte der Datenverkehr auch manipuliert werden.

Eine Einschränkung bei der Implementierung einer solchen Proxy-Software ist, dass bereits vor der Installation der Software die Application IDs der emulierten Karten in XML-Dateien festgeschrieben werden. Das Android-Betriebssystem leitet dann die APDUs, die zur entsprechenden Software gehören, an diese weiter. Falls mehrere Apps die gleichen AIDs beanspruchen, dann kümmert sich das Betriebssystem um die Konfliktbehebung, z.B. indem beim User nachgefragt wird, welche App jetzt zuständig ist. Durch diese Mechanismen ist es nicht möglich, einen allgemeingültigen Proxy für alle Karten zu schreiben, sondern die einzelnen Kartentypen müssen vor der Installation festgelegt werden.

Mit dem Versuchsaufbau in Abbildung 4 gelingt es uns, einen Man-in-the-Middle-Angriff in

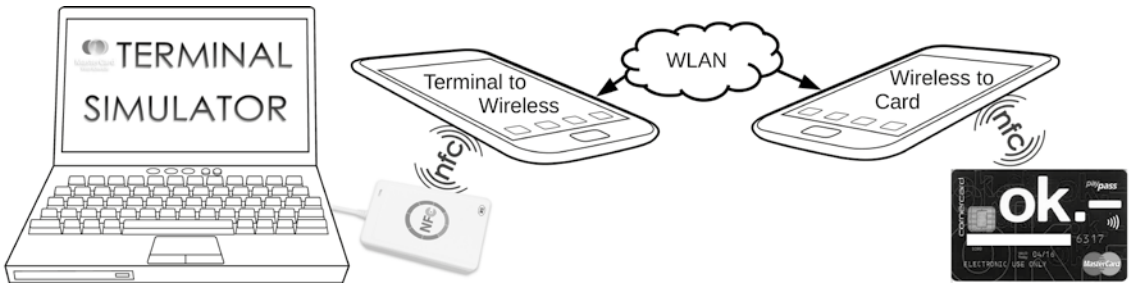

Abbildung 4: Versuchsaufbau zum Testen der Implementierung

der Praxis umzusetzen. Verwendet wird ein LG Nexus 4 und ein Sony Z1 compact; auf beiden Geräten ist Android 4.4.4 installiert. Als Terminal wird im Versuchsaufbau ein Windows-Computer mit der Software *Terminal-Simulator* von MasterCard [MCW] und einem ACR122 NFC-Kartenleser verwendet.

Eine beobachtete Schwierigkeit bei der Transaktion ist die Stabilität der Verbindung zwischen dem „Wireless to Card"-Smartphone und der daran gehaltenen Karte. Während erste Versuche fehlschlugen, konnte durch Trial-and-Error eine Kombination aus einer PrePaid-MasterCard und dem Sony Z1 compact als stabilste Variante ermittelt werden. Bei anderen Karten und Smartphones entsteht oft das Problem, dass die Verbindung zwischen dem „Wireless to Card"-Smartphone und der Karte abbricht, sobald auf der Karte kryptografische Operationen ausgeführt werden. Aufgrund der höheren Leistungs- und damit Stromanforderung der Karten in diesem Moment, gehen wir davon aus, dass die Speisung der Karte über Induktion das Problem ist.

In einem Feldversuch bei einem Detailhändler konnte nachgewiesen werden, dass das Vorgehen nicht nur unter Laborbedingungen funktioniert. Die im Geschäft zwischen dem *Point of Sale Terminal* und der Kreditkarte übertragenen Daten stimmten – bis auf die für jede Transaktion einmaligen kryptografischen Daten – mit den im Versuchsaufbau ermittelten überein.

Der im EMV-Abschnitt dieses Artikels enthaltene Datenmitschnitt ist aus diesen Transaktionen entstanden.

Auslesen von persönlichen Daten

Während der Transaktion werden im Schritt *Read Application Data* diverse Informationen auf der Karte durch das Terminal ausgelesen. Darunter befinden sich mindestens die *Primary Account Number* (PAN, die Kreditkartennummer) und das Ablaufdatum. In manchen Fällen wird zusätzlich noch der Name des Besitzers übertragen, wobei dieser nicht auf allen Karten gespeichert ist. Das Auslesen dieser Daten ist nicht an eine Transaktion gebunden – sobald also das richtige Applet ausgewählt ist, können diese Daten von der Karte gelesen werden.

Damit liegen einem Angreifer, der unabhängig von einer Transaktion eine NFC-Verbindung mit einer Kreditkarte aufbauen kann, genügend Daten für eine Online-Bestellung vor, falls der Händler keine weiteren Sicherheitsfeatures implementiert. Von den Kartenanbietern werden zwei weitere Sicherheitsfeatures angeboten: das Überprüfen des *Card Verification Code* (CVC oder auch CVV für *Card Verification Value*) und ein Online-Passwort (3-D). Einige Online-Händler entscheiden sich jedoch dazu, diese Sicherheitsfeatures nicht zu implementieren, um den Kunden den Einkauf möglichst hürdenlos zu gestalten. Damit tragen die Händler allerdings selbst das Risiko für Kreditkartenmissbrauch. Nur durch den Einsatz eines zusätzlichen Sicherheitsfeatures wird das Risiko vom Händler auf die Bank bzw. den Kreditkartenbesitzer abgewälzt.

Der CVC ist eine Nummer, die bei den meisten Kredit- und Prepaidkarten auf das Unterschriftenfeld gedruckt ist. Diese Nummer (bei MasterCard als CVC2 bekannt, bei Visa als CVV2) ist bloss dem Kreditinstitut bekannt und darf von Händlern nicht zwischengespeichert oder bearbeitet werden. Dies soll ermöglichen, dass bei jeder neuen Bestellung eine Überprüfung stattfindet, ob die Karte physisch vorliegt und somit dieser Code abgelesen werden kann – eine durchaus fragwürdige Verifikation, da ein dreistelliger, optisch lesbarer statischer Code bei einem Diebstahl keine grosse Hürde darstellt. Einzelne Händler wie z.B. Amazon, die Kreditkartendaten von Kunden zwischenspeichern und ein 1-Click-payment ermöglichen, implementieren diese Methode nicht, damit die Kunden ohne Bedenkzeit und ohne Aufwand eine Bezahlung tätigen können.

Mit *3-D Secure* steht Online-Händlern eine weitere Möglichkeit zum Schutz vor Kreditkartenmissbrauch zur Verfügung. Bei diesem Verfahren, das in Europa vor allem unter den Namen *MasterCard SecureCode* und *Verified by Visa* bekannt ist, wird vom Käufer bei einer Onlinetransaktion zusätzlich auf einer zur Kreditkartenfirma gehörenden Website ein Passwort eingegeben. Auch hier gilt, dass viele Onlinehändler auf dieses Sicherheitsmerkmal verzichten, um den Kunden beim Bezahlen möglichst wenige Hürden in den Weg zu legen.

```
public class TerminalToWirelessProxyServiceSimplified extends HostApduService {
   private Socket socket;
   private ObjectInputStream ois;
   private ObjectOutputStream oos;

   @Override
   public byte[] processCommandApdu(final byte[] requestApdu, Bundle extras) {
      new Thread(new Runnable() { /* Not thread-safe in this implementation */
         public void run() {
            if (!isConnected()) { connect(); }
            if (isConnected()) {
               try {
                  /* transmit data to the server, and send response to the terminal. */
                  oos.writeObject(requestApdu);
                  sendResponseApdu((byte[]) ois.readObject());
               } catch (Exception ex) { /* ... */ }
            }
         }
      }).start();

      /* If null is returned here, the function sendResponseApdu(byte[]) can be used to answer
         the request. Here this happens in a separate thread to allow networking. */
      return null;
   }

   @Override
   public void onDeactivated(int reason) {
      disconnect();
   }
   /* ... */
}
```

Listing 6: Ausschnitt aus dem Terminal-to-Wireless-Service des Man-in-the-Middle-Systems (Host Card Emulation Service)

Da bei Online-Bezahlungen der Name auf der Kreditkarte nicht überprüft wird und die beiden von den Kartenausgebern angebotenen Sicherheitsmassnahmen bei Händlern oft nicht implementiert werden, reicht zum Bezahlen oftmals die PAN sowie das Ablaufdatum; zwei Eigenschaften der Karte also, die per NFC auslesbar sind und die bei jeder Transaktion von der Karte an das Terminal übertragen werden. In der „Wireless to Card"-Komponente des Man-in-the-Middle-Angriffs werden die Kartennummer und das Ablaufdatum nach Abschluss der Transaktion in einem Popup dargestellt.

Angriffsvektoren

War es bisher für das Auslesen der Kartendaten nötig, physischen Zugriff auf die Karte zu erhalten, sei es per Skimming oder einer kurzen Entwendung, so ist es jetzt möglich die Daten aus einigen Zentimeter Entfernung auszulesen, z.B. im Gedränge am Bahnhof. Durch diese Vereinfachung des Datendiebstahls in Kombination mit dem Verhalten vieler Online-Händler, die zur Vereinfachung des Bezahlvorgangs auf zusätzliche Sicherheiten verzichten, ist mit einem Anstieg des digitalen Missbrauchs von Kreditkarten zu rechnen. Deshalb wird es auch in Zukunft wichtig sein, die Abrechnung auf verdächtige Zahlungen zu überprüfen – auch oder gerade weil die Haftung für missbräuchliche Zahlungen bei verantwortungsvollem Umgang mit der Karte beim Händler oder beim Kreditkarteninstitut liegt.

War bei der kontaktbehafteten Bezahlung für einen Man-in-the-Middle-Angriff das Einführen eines Gerätes in den Kartenslot eines Terminals notwendig, so kann dieser nun auch mit zwei Smartphones ausgeführt werden. Durch das Aufkommen der Möglichkeit, direkt mit Smartphones zu bezahlen, ist dieses Vorgehen absolut unauffällig. Bereits bevor diese Möglichkeit durch Apple Pay grössere Bekanntheit erlangte, wurden wir zwar von Mitarbeitern beim Detailhandel angesprochen. Diese zeigten sich allerdings eher hilfsbereit beim Versuch, mit dem Handy zu bezahlen, als misstrauisch. Da für kleine Beträge bei der kontaktlosen Zahlung keine PIN erforderlich ist, kann so eine unerwünschte Abbuchung vorgenommen werden, indem ein Angreifer die APDUs vom Terminal über das Internet an ein Smartphone weiterleitet, das mit der Kreditkarte eines ahnungslosen Opfers kommuniziert.

Der Online-Teil der Transaktion, der erst nach dem oben beschriebenen Kontakt zwischen Terminal und Karte ausgeführt wird, kann auch verzögert ausgeführt werden. Im Artikel „Harvesting High Value Foreign Currency Transactions from EMV Contactless Credit Cards without the PIN" [EAF14] beschreiben die Autoren von der School of Computing Science der Newcastle University zwei Sicherheitslücken: Einerseits wird aufgrund eines Implementations- oder Provisionierungsfehlers bei VISA-Karten aus dem vereinigten Königreich bei Transaktionen in Fremdwährungen das Limit für Transaktionen ohne PIN-Eingabe nicht über-

prüft. Dadurch können Beträge bis zu 999 999.99 in der entsprechenden Währung ohne Autorisierung ausgegeben werden. Andererseits beschreiben sie auch das Vorgehen, dass ein Smartphone ein Terminal simuliert und Transaktionen mit Kreditkarten durchführt. Die auf dem Smartphone gespeicherten Transaktionsdaten können anschliessend zu einem beliebigen Zeitpunkt über das Internet mit einem gültigen Händler-Account bei einer Bank platziert werden, um so Geld von den Kreditkarten auf das eigene Konto zu übertragen. Zur Schadensbegrenzung ist bei einigen Kreditkarten ein „Offline-Limit" eingebaut, d.h. nach einer bestimmten Anzahl Transaktionen oder einem bestimmten Betrag ist eine Online-Buchung erforderlich. Auch dieser Angriff ist nur möglich, da bei kleinen (oder durch den Implementierungsfehler bei VISA UK auch grossen) Beträgen keine PIN-Eingabe nötig ist.

Der Nachteil der kontaktlosen Kreditkarte gegenüber der kontaktbehafteten ist es also, dass unbemerkt mit dieser kommuniziert werden kann und nicht immer eine Transaktionsbestätigung durch den Kartenbesitzer notwendig ist. Wenn die Karte jedoch in einem Mobiltelefon gespeichert wird, so ist der Zugriff über die NFC-Schnittstelle nur möglich wenn der Bildschirm aktiviert, eine entsprechende Applikation gestartet worden ist oder gar (wie bei ApplePay) das Auslesen der Kartendaten durch eine zusätzliche Autorisierung (wie ein Fingerprint) freigegeben worden ist. Während also der Zugriff über die kontaktlose Schnittstelle durch Smartphones besser geschützt ist, so besteht bei diesen das zusätzliche Risiko von Malware. Bei einer reinen Software-Lösung kann eine entsprechend programmierte Schadsoftware den privaten Schlüssel der Karte auslesen und versenden. Damit ist es dann möglich, mittels eines anderen Smartphones eine Kopie der simulierten Kreditkarte zu erstellen, um Zahlungen mit den erbeuteten Daten auszulösen.

Eine z.B. bei Google Wallet verwendete Technik zum Schutz des privaten Schlüssel ist es, ein Secure-Element auf dem Telefon zu verwendet. Dieses Secure-Element funktioniert dann gleich wie eine SmartCard, führt also eigene kryptografische Operationen durch und beantwortet APDUs selbständig über die NFC-Schnittstelle des Smartphones, wenn der Bildschirm bzw. die Applikation aktiviert ist. Michael Roland hat eine Relay-Attacke beschrieben, in der – statt über die NFC-Schnittstelle auf eine NFC-Kreditkarte – direkt von der Smartphone-Software aus auf das Secure Element eines Android-Geräts zugegriffen wird [ROL13]. Damit ist der Vorteil verloren, dass der Zugriff auf die Karte nur durch eine Userinteraktion am Smartphone hergestellt werden kann, die Malware kann jederzeit per Netzwerk aktiviert werden.

Fazit

Mit der Möglichkeit kontaktlos bezahlen zu können haben sich gegenüber der kontaktbehafteten Bezahlung per Karte neue Sicherheitsprobleme geöffnet. Risiken, die bereits mit Kontaktkreditkarten bestehen, können dank Kommunikation aus kurzen Distanzen auch ohne Entwenden von Karten oder manipulierte Terminals ausgeführt werden. Dies in Kombination mit der Möglichkeit, kleine Beträge ohne PIN-Eingabe oder Unterschrift bezahlen zu können, führt zu einem gesteigerten Risiko des Kreditkartenmissbrauchs.

Durch die Zahlmöglichkeit mit dem Smartphone, und damit der Ablösung von kontaktlosen Kreditkarten, ist das Risiko gebannt, dass ständig per NFC auf die Karte zugegriffen werden kann: Erst durch Aktivieren des Bildschirms oder sogar einer Benutzerauthentifizierung auf dem Smartphone wird die Zahlung ausgelöst. Dank spezieller Protokollimplementierungen können auch persönliche Daten besser vor manipulierten Terminals geschützt werden. Dafür existiert das Risiko, dass Smartphone-Malware einen anderen Zugangskanal zu der Karte hat: Bei ungeschützter Speicherung des privaten Schlüssels auf dem Smartphone ist es möglich, direkt eine Kopie der Karte anzulegen. Bei einem Secure-Element, das die privaten Schlüssel auf dem Smartphone sicher speichert, besteht implementationsabhängig das Risiko, dass aus dem Betriebssystem mit diesem kommuniziert wird, und so die APDUs eines Terminals über das Internet weitergeleitet direkt an dieses gesendet werden können.

Referenzen

[EAF14] M. Emms, B. Arief, L. Freitas, J. Hannon, A. van Moorsel: Harvesting High Value Foreign Currency Transactions from EMV Contactless Credit Cards without the PIN. http://homepages.cs.ncl.ac.uk/budi.arief/home.formal/Papers/CCS2014.pdf

[ELB] EMV Lab. http://emvlab.org

[FHMM11] L. Francis, G. Hancke, K. Mayes, K. Markantonakis: Practical Relay Attack on Contactless transactions by Using NFC Mobile Phones. https://eprint.iacr.org/2011/618.pdf

[HCE13] Android Developer Documentation: Host Card Emulation. http://developer.android.com/guide/topics/connectivity/nfc/hce.html

[MCW] Master Card Worldwide: Terminal Simulator. https://www.terminalsimulator.com/

[OSC10] Omar S. Choudary: The Smart Card Detective, a hand-held EMV interceptor. http://www.cl.cam.ac.uk/~osc22/docs/mphil_acs_osc22.pdf

[ROL13] M. Roland, J. Langer, and J. Scharinger: Applying Relay Attacks to Google Wallet. In: Proceedings of the 5th International Workshop on Near Field Communication (NFC 2013), pp. 1–6, Zurich, Switzerland, Feb. 2013.

[TVR] TVR-Decoder: Will Currie http://tvr-decoder.appspot.com/t/home

[UTC12] University of Texas at Austin IEEE Communications Society Student Chapter: NFC Proxy. http://www.youtube.com/watch?v=Yjfc60LGjik

Android Best Practices to Improve Battery Efficiency

Multi-core CPUs, motion sensors and multiple wireless radios all draw a significant amount of power which make a regular battery recharge a necessity. Applications, which extensively use the available hardware, reduce the battery runtime severely. Continuous motion sensor recording excessively stress a smartphone's CPU preventing it from entering a deep sleep state. Android 4.4 introduces a new batch-mode for sensors data to keep the CPU asleep for longer periods of time by delaying the reporting of new data from the sensors. The same technique is applied when writing to flash memory. When using a content provider to store data write amplification occurs, which affects writing performance negatively. The longer write times due to the reduced writing speed decrease the battery efficiency even more. A similar batch-oriented strategy reduces the occurrence of write amplification. In return, the reduced writing time improves the battery efficiency.

Chris Yereaztian, Jürg Luthiger | juerg.luthiger@fhnw.ch

Smartphones have evolved at such a rapid pace that they are capable of rivalling basic desktop computers regarding computational power. This trend greatly stimulates the development of mobile software, which in return promotes smartphone sales to an extent that they surpass PC sales [1]. However, battery technology has not caught up with the advancements in chip making. Dual-core CPUs are common amongst smartphones and the additional hardware packed into smartphones such as GPS, Wi-Fi, 3G and 4G radios all take its toll on the battery. A regular overnight recharge of the smartphone has become the norm due to the limited capacity of the batteries. Slow advancements in increasing the capacity combined with more powerful hardware can severely affect the battery efficiency. The applications running on a smartphone, which make extensive use of the hardware, are therefore at fault for the diminishing battery efficiency. In order to reduce the strain applications put on the hardware, developers need to understand the inherent restrictions of a mobile environment and the implications it has on their applications.

Battery Drain

In particular, applications using hardware features such as the motion sensors as well as the built-in flash storage significantly drain the battery. When an application requests access to the motion sensors it does so by using the sensor framework to subscribe to one of the sensors. In return, the sensor continuously publishes raw sensor data to the application, which keeps the CPU busy and prevents it from entering a deep sleep state. Writing the acquired raw data to the built-in storage leads to the effect of write amplification where more data needs to be written into the flash memory than actually specified due to the limited write capabilities of flash.

Since the use of the motion sensors is the underlying cause of the battery drain, on Android platforms a new batch-mode for listening to sensors data is introduced with version 4.4. It allows sensors to delay the reporting of new data to the application. The CPU is kept asleep for longer periods of time, which positively affects the battery efficiency.

A similar batch-oriented approach can also be applied to reduce the occurrence of write amplification. Writing a single record to a *Content Provider* in Android can trigger multiple writes on the flash storage that increase the power consumption. Instead of using one database transaction for writing each record, multiple records can be combined to a single transaction eliminating the transactional overhead. Storing data in larger junks less often compared to smaller ones more often improves the battery efficiency since the additional write operation necessary are kept at a minimum. As shown later in this article the GreenDAO framework supports this strategy when writing multiple records into a database.

Another aspect that has not been mentioned yet, but greatly influences the battery efficiency as well, is the wireless radio. Sending data to the Internet over UMTS consumes the most amount of power compared to other components as shown in Table 1. Applications that run services in the background, which access the Internet, keep the wireless radio awake for longer periods of time. Furthermore, the state machine of the wireless radio negatively amplifies the wake lock on the radio. When an application finishes a transmission the radio stays active for another 5 seconds before transitioning into a lower power state mode because re-activating the radio is bound to a delay. After another 12 seconds without any transmission the radio goes into a deep sleep state [2]. On the assumption that typically multiple applications are running in the background the radio

Action	Nexus 4 [mAh]	Nexus 5 [mAh]
UMTS Download	1339	1073
UMTS Upload	1410	1033
UMTS Call	983	637
UMTS Standby	18.3	10.4
WiFi Download	1158	549
WiFi Upload	475	488
GPS Searching	550	263
GPS Standby	0.4	0.7
NFC Standby	-	4
Sensors	751	487
Display (max)	310	567

Table 1: Power usage of different components measured with the TrepN profiler

is left active. Since not all applications transfer data to the Internet at the exact same moment in time the radio never enters the deep sleep state and therefore drains the battery.

In this article we discuss in more detail the battery drain caused by motions sensors and flash storage. The effects of wireless radio are omitted since the complexity of the interaction between the radio and the software legitimates its own analysis.

Sensor Batching

The built-in motion sensors in mobile phones such as the Nexus 4 and 5 draw a significant amount of power when the sensors are actively running, about half as much as the wireless radio (Table 1). The sensor framework provided by Android is built upon four distinctive classes and interfaces. The *SensorManager* provides access to the sensor services. It contains methods for directly querying the available sensors and registering sensor listeners. The *Sensor* class is used to create a specific instance of a sensor to determine its capabilities. The *SensorEvent* class is used by the system to create events, which include the raw, time stamped data of the sensor and the type of sensor that generated the event. The *SensorEventListener* interface defines two callback methods which the system calls when the values of a sensor or the accuracy of a sensor itself changes.

The best practice for accessing the motion sensors, before the introduction of the previously mentioned batch mode, used to be to limit the time window the application is actively subscribed to the sensors to a minimum. The activity class is a critical component of an application and transitions through multiple states in its lifecycle. When an application transitions to the state where it is actively in the foreground and visible to the user, the *onResume()* callback method of the corresponding activity is called. This is the recommended place to subscribe to the motion sensors. In contrast, when an application loses the focus or is partially covered due to another dialog, the activity is paused and the *onPause()* callback method is called. Even though the application may return to its running state the Android documentation recommends unsubscribing the sensor listeners to limit the unnecessary battery drain.

However, this approach cannot be applied to an application that is constantly recording the sensor data. For example, a pedometer requires constant access to the motion sensors to determine the amount of steps and detect possible false positives. With the existing method the applica-

Figure 1: left) Electronic interface to read out power consumption of a smartphone. right) Nexus 4 with an open back cover for connecting measurement equipment.

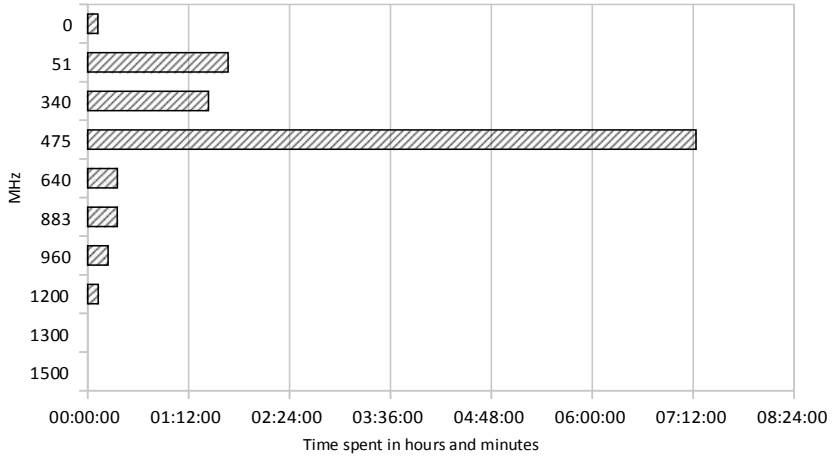

Figure 2: CPU usage of a Nexus 5 in non-batch mode

tion would severely drain the battery. The new batch-mode introduced with Android 4.4 permits constantly listening to the sensors without significantly increasing the battery consumption. The batching refers to the mechanism of bundling multiple values from the sensor before sending a new event to the sensor event listeners. The Android sensor framework support this by adding an additional parameter to the *registerListener()* method that allows the caller to specify the maximum latency in seconds before a batch needs to be send to the listeners [3]. If this parameter is set to zero, batch processing is completely disabled. The significant power savings, as seen in Figure 2, are based on preventing the *System on Chip* (SoC) of waking up for each receiving sensor event. Multiple events can be grouped and processed together while each of them retains their own individual timestamp. The batching is done in hardware using FIFO queues, which temporarily hold the sensor events before sending them through hardware abstraction layer to the system [4]. The oldest events in a queue will be dropped if there is not enough space available to accommodate new incoming events from the sensors.

The sensors in a smartphone are divided into two categories: the wake-up sensors and the non-wake-up sensors. Sensor events from wake-up sensors are stored separately in a wake-up sensor queue to ensure that the data is delivered regardless of the SoC's current state. The driver of the wake-up sensors achieves this by keeping a wake-lock for at least 200 ms to ensure that the new event is delivered to an application. As a result the wake-up sensors, as their name implies, will wake-up the SoC to deliver the events before the specified maximum allowed reporting latency has elapsed [5].

In contrast, non-wake-up sensors do not prevent the SoC from entering in its sleep state and more importantly will not wake up the SoC to report new sensor events. The driver of the non-wake-up sensors does not hold a wake-lock. Therefore, the application is responsible for keeping a partial wake-lock if it wants to receive new events from non-wake-up sensors while the screen is off. The events in the FIFO queues will be delivered to the application as soon as the SoC returns from its sleep state.

The batching cannot be emulated in software and needs to be implemented in hardware. Since

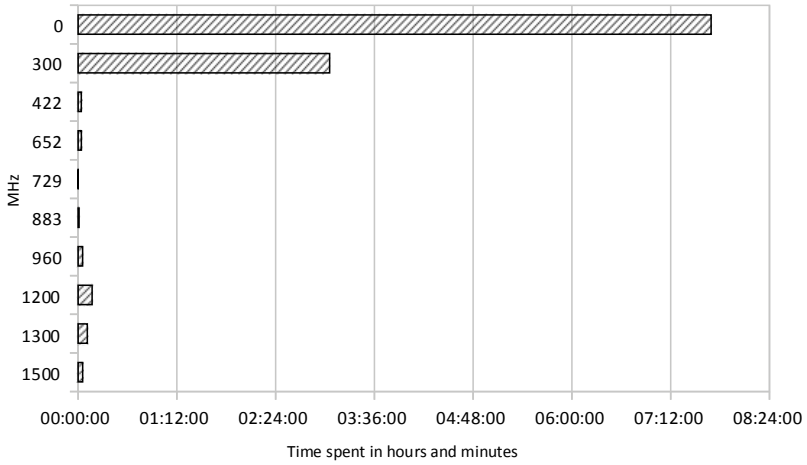

Figure 3: CPU usage of a Nexus 5 in batch-mode

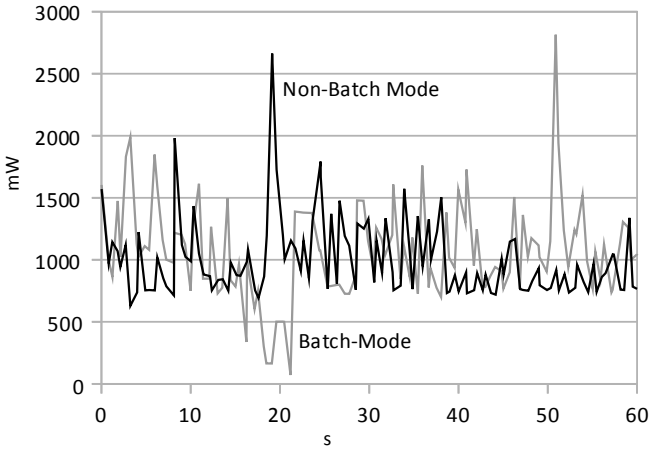

Figure 4: Battery usage of a Nexus 4 in non-batch and batch-mode

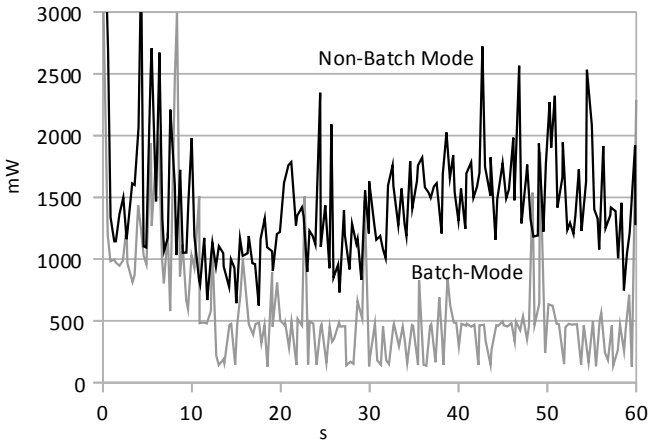

Figure 5: Battery usage of a Nexus 5 in non-batch and batch-mode

the goal of batching is too reduce the battery drain caused by sensors waking up the SoC, it definitely cannot be implemented using the SoC itself. Therefore, a separate hardware chip that provides and manages the FIFO queues is required to keep the SoC in suspend mode during batching. Older devices such as the Nexus 4 are not able to take advantage of this new feature in Android 4.4 as they lack the necessary hardware chip for the queue. Registering a sensor listener with the *maxBatchReportLatency* set to other than zero will silently be ignored. The sensor listener will receive the events as if it had registered the listener with batching disabled.

Sensor Batching Tests

The effectiveness of the proposed practices is shown with a series of tests on two different Nexus smartphones, Nexus 4 and 5. Both devices are based on nearly the same Qualcomm Snapdragon platform and equipped with same model of sensors. However, the Nexus 5 also includes the required separate hardware chip to manage the formerly discussed FIFO queue. In order to create an identical testing environment on both devices, the latest stock Android 4.4 image (latest at the time of writing) is used without any modifications to the kernel. Since Android supports running applications concurrently in the background, the same number of applications is installed on both devices. This ensures that none of the running background processes will influence the measured results because if one of process would indeed affect the battery drain, it would do so on both devices.

Diagnosing the battery drain requires direct access to the power management on a smartphone. On all Qualcomm Snapdragon based smartphones such as the Nexus 4 and 5 the kernel provides access to the power management. The *TrepN Profiler* [6] is a plugin for Eclipse that uses kernel functions to read out the statistics from the internal power management chip. This enables the accurate profiling of the overall CPU usage and frequency, memory as well as network usage. To measure the overall power consumption of a Nexus device the back cover needs to be opened because the battery with the connectors (marked in Figure 1b) is sealed inside the body. The electronic interface (Fig. 1a) connects to a debug port on the battery connector (4-pin socket instead of the usual 2-pin plus and minus pole socket). To minimize the effect of measuring errors on our test results, each test is run three times and the average of all three runs is taken as the actual measurement value. If one measurement of those three runs differs from all the others by more than 10% then that specific run is repeated and the current measurement is discarded.

As already mentioned the hardware-backed FIFO queues require a separate chip that takes over the management of the queues. Our test of continuous recording sensors shows a clear difference in CPU usage between batching disabled and enabled on a Nexus 5. The CPU in the Nexus 5 is kept asleep for longer periods since no interrupts from the sensor are waking up the SoC. The 0 MHz bar in the charts (Fig. 2 and 3) represents the CPU in deep sleep state (the 0 MHz is just the interpretation of the *TrepN* profiler). The time axis represents the time the CPU was running at the specific frequency. With batching disabled the CPU spends most of its time running at 475 MHz for processing sensor data even if the application is idling, as seen in Figure 2. In contrast, Figure 3 shows that enabling batching keeps the CPU asleep for longer periods of time (Figure 2 and Figure 3 use different scaling since the CPUs dynamically scale the clock up and down depending on the workload). Figure 4 and 5 confirm that the different CPU states affect the battery drain. The battery usage on a Nexus 4 shows no difference between measuring sensors in non-batch and batch mode in Figure 4. Since the Nexus 4 does not have the required hardware chip the Android's sensor framework will automatically fall back to non-batch mode. In comparison, the Nex-

us 5 in Figure 5 does show that the battery usage in batch-mode is drastically cut by more than one third. Logically, due to the lower CPU usage and the fewer interrupts caused by wake-ups from sensor events, the overall battery consumption is reduced.

Flash Memory

So far we have shown that accessing the sensors contribute significantly to the battery drain. The presented solution of batching sensor events together has proven to positively affect the battery drain issue. However, the sensors are not the only component in a smartphone influencing the power consumption in a significant manner. Depending on its use, the flash storage increases the battery drain as well. The typically used flash memory (eMMC) in mobile phones suffers from an unwelcome effect of write amplification where more data is written to the flash than the application actually committed to [7]. This is primarily due to the nature of flash memory where data cannot be directly overwritten compared to traditional hard disk drives [8]. The severity of write amplification is linked to the storage provider used in an application. A storage provider on a mobile phone should support fast reading and writing of data while using a minimal amount of power. Furthermore, it should facilitate the processing (searching, sorting, and filtering) and sharing of data between applications on the phone while making the files unattainable for direct modification from the outside to secure their integrity.

The Android storage framework provides the following mechanism to store data in an application: *Shared Preferences* represent a key-value pair store, which holds primitive data types. It is designed to store internal application preferences rather than user object-data. A *Content Provider* is a mechanism that manages the access to a set of structured user data. The provider encapsulates the data by providing a common CRUD interface that allows easy access to the data from within the application as well as third party applications. The Android framework also provides read/write access to the underlying file system provided by the Linux kernel. Applications can use the file system to store user data and preferences in files regardless of data structure and format being employed.

Shared Preferences are designed for persisting preferences that can be retrieved and set using a string as a key to identify the associated value. The critical issue with *Shared Preferences* is that they are application specific and therefore cannot be used to share data. Files suffer from the same problem. All applications on Android run in a sandboxed environment where each application is run with a separate user account. The associated files are kept private and no other user respectively application can access these files [9].

The *Content Provider* mechanism is a set of methods and structures designed to separate the raw data from the aggregated complex data at runtime. For example, the *Contact Provider* combines information about the contacts from multiple sources such as a SIM card, contacts from a Google account, application specific contacts etc. The provider itself can implement the usage of the storage in two ways:

- Firstly, a file-centric approach where the data normally goes into files which are stored in the application's private space.
- Secondly, a structured data approach where the data is placed into a database, an array or a similar structure by mapping the data into a set that is compatible with rows and columns. In general, any type of storage can be used, but a common way to store this type of structured data is to use an *SQLite* database, because Android offers built-in support for *SQLite*. A substantial part of the writing process is hidden from the developer when a single record is put into a database. The commit call for the database immediately returns and the developer assumes that the data has already been written to the disk and moves on.

It is inherently difficult to determine which components of the process are the least efficient and affects a phone's battery the most. To identify the factors that influence the power consumption, a detailed understanding of how embedded Multi-Media-Card (eMMC) storage works is necessary.

Write Amplification

General flash-based storage found for example in solid state drives separate the NAND flash chips from the controller. The controller is a critical component as it handles the mapping between disk-based track and sector geometry and the flash-based cell geometry. Due to the small device requirements of smartphones, the controller and NAND flash memory are contained within one package [10].

When writing data to flash storage, an undesirable but unavoidable issue is the occurrence of write amplification where the actual data written to the flash is a multiple of the data that the host requested to be written. The root cause of write amplification is that individual pages of memory can be written to empty flash memory cells but the pages can only be erased in larger units, called blocks. If a block contains invalid and valid pages, then the eMMC controller must first read in all the valid pages of a block it wants to write new data into. After caching those valid pages it will invalidate all pages in that specific block and finally write both the old and updated pages back to the block.

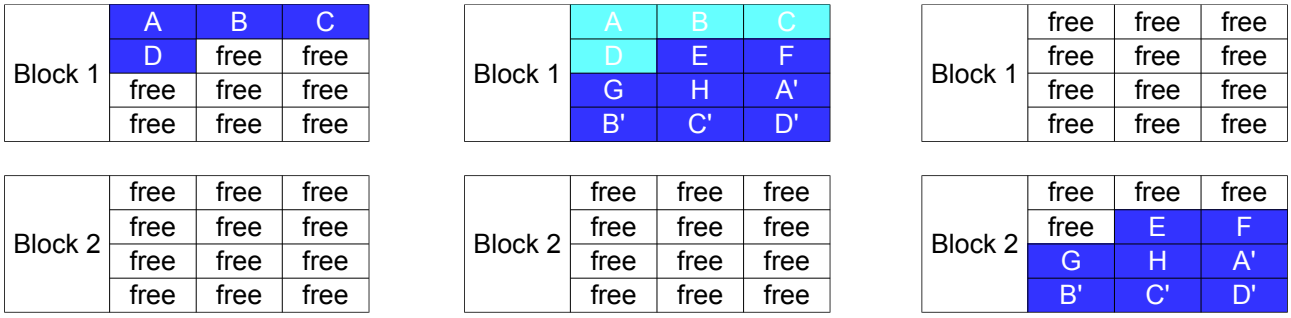

Figure 6: left column) Pages written to free memory cells in a block. middle column) Invalidating old pages (stale data) and replacing them with new ones in a block. right column) Cleaning up stale data and writing old as well as new data back to the block.

The example depicted in Figure 6 illustrates the issue quite well:

1. In the left column of Figure 6 four pages (A–D) are written to block 1. Individual pages can be written at any time if they are currently free respectively erased.
2. In middle column of Figure 6 four new pages (E–H) and four replacement pages (A'–D') replacing (A–D) are written to block 1. The original pages (A–D) are invalidated (so called stale data) and cannot be overwritten with new data until the whole block has been erased.
3. In order to overwrite pages with stale data (A–D) the remaining valid pages in block 1 (A'–D' and E–H) are read, cached within the controller or directly written to another block (right column in Figure 6). Now, the controller is able to completely erase block 1 which resets the memory cells so new data can be written into it.

The undesired increased number of writes occupies bandwidth to the flash storage and hampers the random write performance severely. When the storage is relatively empty write amplification doesn't occur since there are enough fresh empty cells the controller can use. However, when a large amount of data has already been written to the flash memory and only few empty cells are available, write amplification occurs. The controller needs to shift valid pages to other blocks in order to be able to clear all invalid pages, as seen in the former example. The I/O performance of Google's Nexus 7 (2012 Edition) suffers from this issue where the device's subjective performance slows down after months of use, which leads to an inconsistent user experience.

Since Android 4.3 the negative effect of write-amplification has been alleviated, because Google has enabled the support for *TRIM*, a mechanism that trims blocks that are not used in the file system. Briefly summarizing, *TRIM* allows the OS to tell the eMMC controller that a block is no longer in use and ready for garbage collection. It is important to mention that deleting a file or a record in a database is not actually communicated to the eMMC controller. Even though the space is freed up in the file system the controller still treats the block with the pages as containing valid data that cannot be purged. The controller is forced to move pages that are invalid but still treated as valid pages in the block and hence needs additional power. Exactly quantifying the additional battery drain is difficult, because file system operations are handled by the Linux kernel and applications do not have a direct way of managing those from user space to observe when exactly write amplification occurs. However, when writing records into a *Content Provider* backed by a database the hypothesis that a correlation between battery consumption and the occurrences of write amplification exists can be made. The higher the write amplification the more power the smartphone will draw from the batteries since it requires more time to complete the writing process. To test this hypothesis 10000 objects consisting of sensor event values are inserted into a *Content Provider* while the execution time and battery consumption are measured. Three distinctive approaches for inserting the records into the database are used:

- Each single record is inserted separately using the methods provided by a *Content Provider*. No optimizations are done in a *Content Provider* or the insertion call itself.
- A batch-oriented approach is applied. Similar to the solution of the sensors where multiple events are bundled together, multiple records are bundled to a batch for inserting them at once into the database. The idea is to eliminate the transactional overhead as well as preventing possible write amplifications that might occur in the first approach due to repeated writings of smaller data junks.
- GreenDAO, an object-relational mapping (ORM) framework for Android is used for storing the sensor event objects into a database [11]. It promises to put its focus on maximum performance. GreenDAO takes over the responsibility of a *Content Provider* and offers methods to persist objects directly.

In order to minimize measurement errors, again each test is repeated three times and the average execution time of the three runs is used for the comparison. The test results depicted in Figure 7

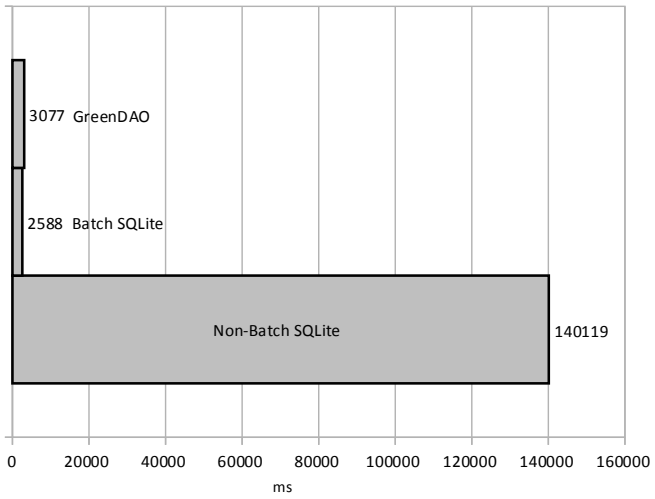

Figure 7: Execution time for write operation into a database

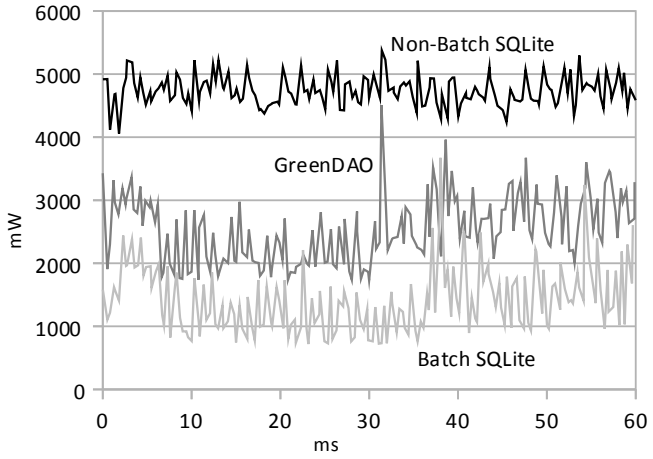

Figure 8: Comparison of battery usage between different storage providers

show a distinctive difference between the batch and non-batch approach of inserting the records into the database. Using a single transaction for inserting 10000 objects takes 2.5 seconds. It is relatively fast in comparison with the non-batch approach that requires 14 seconds. The negligible difference between the ORM framework GreenDAO and batch-insertion is due to the overhead of object-relational mapping. The measured battery usage in Figure 8 reflects the same result. There is a considerable difference between batch and non-batch approaches. The drain is about twice as high for not batching the records. In contrast, the battery consumption only slightly increases from manual batching to using GreenDAO. Since the CPU usage between all three approaches is nearly identical, the correlation can be made that the increased consumption is caused by the amplified writes to the flash memory. The less time a smartphone is spending on data persisting, the more battery can be saved.

Summary

Application development on mobile platforms is still more difficult compared to desktop systems because of the limited resources available on mobile devices. Applying specific techniques considerably improves the battery efficiency as shown in the presented examples. Compared to desktop systems where the abundance of resources eliminates any incentive of optimizing an application, small changes in mobile applications lead to noticeable improvements. Bundling operations in batches has proven to be an effective technique in general. Applied to the sensor framework as well as the persistence mechanism it drastically reduces the power consumption.

References

[1] Rob van der Meulen, Janessa Rivera. Gartner Newsroom Press Release, 2014. https://www.gartner.com/newsroom/id/2791017

[2] Oliver Spatscheck, Alexandre Gerber, Subhabrata Sen. A call for more energy-efficient apps, 2011. http://www.research.att.com/articles/featuredstories/201103/201102Energyefficient?fbid=pvldc4jPUQE.

[3] M. Hidaka. Android 4.4 sensor batching. 2013. http://techbooster.org/android/device/16666/

[4] Android Developers Documentation. Android Sensor Batching. 2014. https://source.android.com/devices/sensors/batching.html

[5] Android Developers Documentation. Suspend mode – Wake-up Sensors, 2014. https://source.android.com/devices/sensors/suspend-mode.html#wake-up_sensors

[6] Qualcomm Technologies. Increase app performance with TrepN profiler, 2013. https://developer.qualcomm.com/mobile-development/increase-app-performance/trepn-profiler

[7] Xiao-Yu Hu, Evangelos Eleftheriou, Robert Haas, Ilias Iliadis, Roman Pletka. Write Amplification Analysis in Flash-Based Solid State Drives. IBM Zurich Research Laboratory.

[8] Cameron Crandall. SSD: Flash Memory and Write Amplification. Kingston Technology. http://www.kingston.com/us/community/articledetail?articleid=17

[9] Nikolay Elenkov. Android Security Internals: An In-Depth Guide to Android's Security Architecture. No Starch Press, p12, p50-59, 2014.

[10] Datalight Technologies. What is eMMC? http://www.datalight.com/solutions/technologies/emmc/what-is-emmc

[11] Vivien Dollinger, Markus Jungiger. greenDAO – Android ORM for SQLite, 2013. http://greendao-orm.com/features/

Software/Hardware Co-design: Crypto MicroCore

MicroCore is a dual stack, Harvard architecture with three memory areas that can be accessed in parallel. One special feature of MicroCore is the possibility to define more complex Forth words by creating new CPU instructions that later can be implemented via VHDL in FPGA hardware. In our project[1] we developed a new version of the MicroCore C-compiler based on lcc, a fully ANSI-C compliant compiler. The compiler generates from C-Code MicroCore instructions for the 1.71 version which are sent to the target hardware with the MicroForth-loader. We built in optimizations for the global stack allocations, first suggested by the group of Chris Bayley at the University of York. We tested our compiler with the BLAKE hash algorithm, implemented both in compiled MicroCore code and with BLAKE-optimized instructions directly coded in hardware via VHDL.

Markus Knecht, Willi Meier, Klaus Schleisiek, Carlo U. Nicola | carlo.nicola@fhnw.ch

MicroCore is a dual stack, Harvard architecture with three memory areas that can be accessed in parallel: data stack (RAM), data memory and return stack (RAM), and program memory (ROM). All instructions without exception are 8 bit wide, and they are stored in the program memory ROM. Due to the way literal values can be concatenated from sequences of literal instructions, all data paths and memories are scalable to any word width as long as the needed address space can be represented. Figure 1 shows the MicroCore's structure [7].

One special feature of MicroCore is the possibility to define more complex Forth words by creating new CPU instructions that later can be implemented via VHDL in FPGA hardware. The synchronous simple design of MicroCore is tailored toward an easy and cheap implementation in FPGA. Version 1.71 of MicroCore (the current specification) introduces a new and better subset of Forth as a machine language especially tailored for cryptological calculations. A simple VHDL interpreter allows the cross compiler to load "CONSTANTS.VHD", which defines hardware features and opcode mnemonics. MicroCore/MicroForth thus evolves to a co-design environment. Every change in the hardware is unambiguously brought forward to the cross compiler. It is guaranteed by design that names of hardware characteristics are spelled the same in VHDL and in Forth, and that they have identical values.

The control part of MicroCore (see Fig. 2) is represented by a decentralized Moore-FSM (Finite State Machine) whose next state is determined by the decoder. Decentralized means that semi-autonomous parts of the data path (i.e. the local- and data-stack) are directly fed by the decoder. Why decentralized? For two reasons: firstly, the logic of the FSM is simpler, and secondly, the VHDL description maps more gracefully into the hardware components.

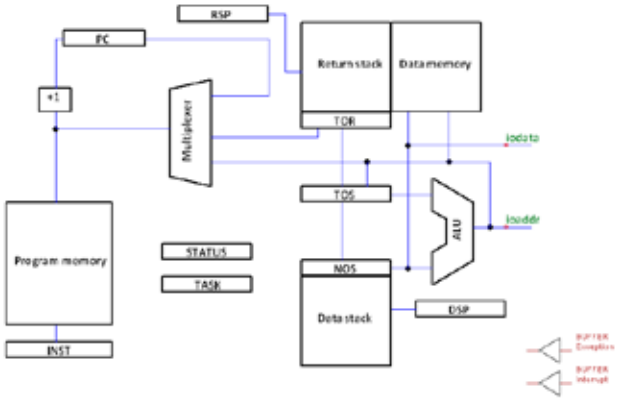

Figure 1: Overview of MicroCore

The data-path (Fig. 3) regulates the flow of operands to or from the two stacks. We have not littered the picture with all the necessary multiplexers in order to better illustrate the underlying structure. TOR, PC, DSP and RSP (Table 1 lists the most important abbreviations) each have a primitive n-bit-adder, because of the increment/decrement and push/pop instructions associated with them. Please, notice that manipulations of DSP, RSP, PC and INST occur simultaneously and can be done within a single clock cycle. All registers, stacks, and internal busses are n-bit wide. Since a MicroCore's instruction is 8 bit long we can store $k = n/8$ instructions per clock cycle in INST, transforming in fact the register INST into a small cache.

The data-path is made up of the data-stack, the ALU and of the data memory and return stack. Registers TOS and NOS feed the ALU. TOR allows not only for a decrement-and-branch instruction that can be nested, but also for complex math instructions like multiply, divide or vector multiply. Adding the TASK register will support a multi-tasking OS with a base-index address-

1 This work was partially funded by the Hasler Foundation Grant no. 12002 "An optimized CPU architecture for cryptological functions".

STATUS	Status register with eight 1-bit flags: C = Carry, OVFL = Overflow, IE = Interrupt Enable, IIS = Interrupt In Service, LIT = Literal, N = Sign flag, ZF/DIV = Zero Flag, TIMES = Counter for TIMES instruction.
INST	Instruction register
TOS, NOS, DSP	Top Of Stack register, Next Of Stack register, Data Stack Pointer. All work together with the data-stack (DS).
TOR, RSP	Top Of Return stack register, Return Stack Pointer. This set of registers works with the return-stack (RS).
PC	Program Counter register
TASK	Register whose content points to the Task Control Block (TAB).

Table 1: Glossary of the abbreviations we use to describe the data path

ing mode into the task control block. Indexed addressing into the return stack gives single cycle access to local variables. The data stack is realized by a single port RAM used as stack under the control of the DSP, and the topmost stack item is held in the TOS register. Typically, the size of the data-stack memory needed will be small enough to fit inside the FPGA.

IO is data memory mapped and the most significant address bit selects the external world when set. In addition, program memory may be mapped into the lower part of the IO address space for von Neumann style read and write access in two cycles during the development phase. If the most significant address bit is not set, data-memory and return stack RAM is selected. The return-stack grows towards lower addresses and typically occupies the upper end of data memory under the control of *Return-Stack-Pointer* RSP. Both data and program memory may use internal FPGA block-RAM as "caches" and therefore, MicroCore can run as a "single chip controller" inside an FPGA without any external memory

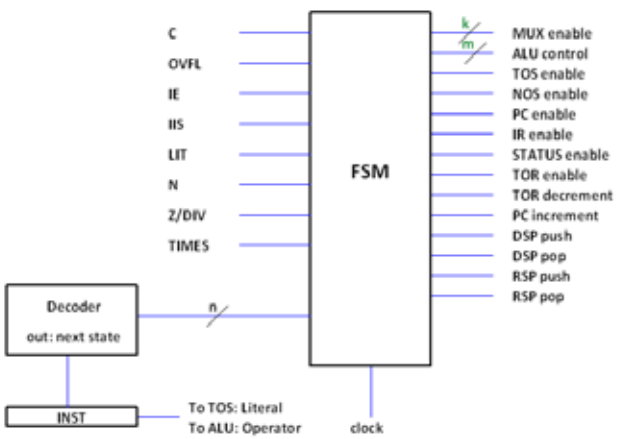

Figure 2: The control unit. n, m, k: number of control lines dependent in general from the width in bits of the data bus and the CPI (Clock Per Instruction).

needs. Several data memory access instructions are available:

- Absolute addressing with the address in the TOS register and a three-bit-signed pre-increment/decrement in the instruction (ld, st);
- Indexed addressing relative to the RSP for local variable access (lld, lst);
- Indexed addressing relative to the TASK register (tld, tst).

After each memory access, the absolute memory address that had been accessed will remain in TOS. Data transfer takes place between the second data-stack element NOS and memory/IO.

The Sequencer generates the program-memory address for the next instruction, which can have a number of sources:

- the PC for a sequential instruction;
- the ALU for a relative branch or call;
- the TOS register for an absolute branch or call;
- the TOR register for a return instruction;
- the INSTruction register for an immediate call (soft instruction);
- the fixed *Interrupt Service Routine* address (ISR) as part of an interrupt acknowledge cycle;
- the fixed *Exception Service Routine* address (ESR) during an exception cycle;
- the fixed *Overflow Service Routine* address (OSR) for a conditional service routine on overflow.

In the code, *uBus* has been defined as a record of signals that are needed in several entities: the data memory and I/O signals, a selection address for the internal registers, an array of all register outputs so as to simplify adding application specific registers. For better and easier code maintenance, instruction decoding and status register bit processing have been centralized since version 1.50 in a single file "uCore.vhd".

In Figure 4 we show the MicroCore development board we used throughout this work.

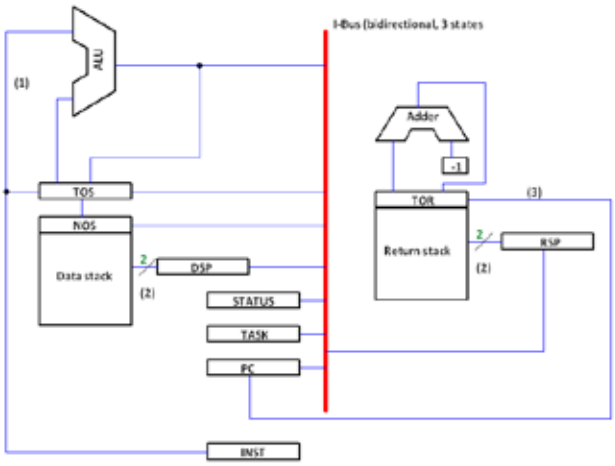

Figure 3: The data path. (1) Because of ++!, ...: 3 bit increment value will be added to TOS; (2) push and pop multiplexer lines; (3) direct way to save next instruction's address during call, int, ...; ALU top input: RSP, NOS, TASK, PC (not all connections are shown in this simplified representation).

Figure 4: The MicroCore development board. The large chip is the Lattice LFXP2-17E-5Q208C FPGA into which we implemented the different versions of MicroCore. Size: 94 x 70 mm.

7 $80	6 $40	5 $20	4 $10	3 $8	2 $4	1 $2	0 $1
Lit/Op	Type		Stack		Group		

Figure 5: The format of an instruction

MicroCore Instruction Set Architecture (ISA)

All MicroCore instructions are 8 bit wide. If the 7th bit is set, the instruction is interpreted as a 7 bit literal (LIT). LIT can be joined to form arbitrary long integers, which are stored in the operand stack. Two LITs build a LITERAL (a 16-bit number in two's complement format). It is a compiler's task to preserve the correct sign when joining many LIT together. The data size is thus independent from that of the instruction. As shown in Figure 5 all instructions are partitioned in fields, whose width in bit remains always constant. The instruction set architecture is based upon the Forth language.

The MicroCore's ISA implements a rich subset of the Forth language. Why, Forth? Forth has evolved in the last thirty years through natural selection in the embedded systems niche, a simple but powerful group of instructions that are well understood and whose side effects are empirically well known. We shall show in Tables 2-4 how to encode the three opcode's format fields.

With the encodings in Tables 2, 3, and 4 we can now synthesize meaningful Forth words, as Table 5 shows. The symbol - in the column LIT means that the previous instruction was an opcode; whereas when the symbol changes to +, that it was a LIT. In this case we get a new LIT on TOS. A * means, as always, don't care.

The regular structure of the three fields (Type, Stack, Group) and the constant bit format of all instructions simplify enormously the complexity of the decoder. However, we hasten to add that the encoding of the Forth words is not entirely satisfactory: for example, we had to use the stack field NONE to push or pop data into the local stack. This represents not only a breach in the orthogonality of the ISA, but more seriously, a supplementary hardware layer in the decoder.

Code	Name	Action
00	BRA	Branches, calls and returns
01	ALU	Binary and unary operations
10	MEM	Data memory and register access
11	USR	Free for application specific extensions. As to version 1.71, 26 user defined extensions are possible.

Table 2: Type field encoding

Code	Name	Action
00	NONE	Dependent on type field encoding
01	PUSH	TOS → NOS → Stack
10	POP	Stack → NOS → TOS
11	BOTH	Dependent on type field encoding

Table 3: Stack field encoding

Code	Binary Operation	Unary Operation	Condition	Register
000	add	not	never	status
001	sub	sl	zero	tor
010	adc	asr	sign	rstack
011	sbc	lsr	carry	local
100	and	ror	pause	rsp
101	or	rol	int	dsp
110	xor	zequ	dbr	task
111	nos	cc	always	flags

Table 4: Group field encoding. The group field determines operations, conditions, and registers in dependence on type field encoding. The different type fields behave as follows: ALU specifies binary and unary operations; BRA specifies the conditions under which a jump is executed; MEM specifies a register name.

For the crypto version of MicroCore we introduced some new instructions:

ROT32: Instead of *times* we define rot32: ROT32 (32b n - 32b')

Label (<name> -), C: Compiles <name> into the dictionary as a constant, which holds the current program memory address. If <name> is the destination of a preceding GOTO, ?GOTO, or CALL, pending forward references will be resolved.

GOTO (<name> -), C: Compiles an unconditional branch to <name>. <name> may be a label or colon definition. GOTO supports forward referencing, i.e. the name of the label or colon definition that follows may be defined later on.

?GOTO (<name> -), C (zero: n -) (sign: n -) (carry: -) (ovfl: -): Compiles a conditional branch to <name>. <name> may be a label or colon definition. ?GOTO supports forward referencing, i.e. the name of the label or colon definition that follows may be defined later on.

- ?GOTO compiles 0<>branch
- 0= ?GOTO compiles 0=branch

Forth	LIT	Implementation	Comments
NOP	*	BRA NONE NEVER	No Operation; 0 → LIT
>R	*	MEM NONE RSTACK	Stack → NOS → TOS → R-Stack
R>	*	MEM BOTH RSTACK	R-Stack → TOS → NOS → Stack
OVER	*	ALU PUSH NOS	
+	*	ALU POP ADD	Stack → NOS<+>TOS → TOS
-	*	ALU POP SUB	Stack → NOS<->TOS → TOS
AND	*	ALU POP AND	Stack → NOS<AND>TOS → TOS
BRANCH	-/+	BRA POP ALWAYS	Absolute 3, relative 2 clock cycles
CALL	-/+	BRA BOTH ALWAYS	Absolute 3, relative 2 clock cycles
EXIT	*	BRA NONE ALWAYS	
L@	*	MEM BOTH LOCAL	LS[RSP + relAddr] → TOS

Table 5: Forth words encoding

- 0< ?GOTO compiles s-branch
- 0< 0= ?GOTO compiles ns-branch
- carry? ?GOTO compiles carry? IF ELSE GOTO THEN
- carry? 0= ?GOTO compiles nc-branch
- ovfl? ?GOTO compiles ovfl? IF ELSE GOTO THEN
- ovfl? 0= ?GOTO compiles no-branch

CALL(<name> -), C: Compiles an unconditional call to <name>. <name> may be a label or colon definition. CALL supports forward referencing, i.e. the name of the label or colon definition that follows may be defined later on.

A synchronized RESET signal resets all registers to zero with the exception of the INST register. Instead, INST loads the code for a NOP [BRA NONE NEVER] and therefore, the instruction whose address is in PC (which has been reset to zero!) will be fetched during the first cycle (which is the NOP instruction).

Exceptions

MicroCore knows only one type of exception; hardware interrupts. They are synchronized with the internal clock and are answered at the end of the actual clock cycle, if the IE bit in STATUS register is set. During the first clock cycle after granting the interrupt request, MicroCore:

1. stores the already correct next instruction's address into PC without any increment, and
2. loads the hard-wired instruction BRA BOTH INT into IR. This call instruction selects the hard-wired address of the interrupt handler.

During the second clock cycle, after granting the interrupt request, MicroCore:

1. executes the instruction BRA BOTH INT. That means two things: transfer of the content of the STATUS register on the Stack and of the content of the PC on the L-Stack; and
2. loads the first instruction of the interrupt handler, since its address was already prepared during the first cycle.

In summary, only the first INT-cycle must be performed by special hardware. The second cycle (INT-instruction) is executed by an instruction which is forced into the INST register during the first interrupt acknowledge cycle. During the STATUS register's transfer on the stack, the IIS bit is automatically set. New interrupt requests are from now on granted, only if we explicitly set IE and reset IIS.

Whenever an interrupt source is asserted, whose corresponding interrupt-enable bit is set in the IE-register, its associated bit in the FLAGS-register will be set and an interrupt condition exists. An interrupt acknowledge cycle will be executed when the processor is not currently executing an interrupt (IIS-bit not set) and interrupts are globally enabled (IE-bit of the STATUS-register set). Please, note that neither the call to the ISR-address nor reading the FLAGS-register will clear the FLAGS-register. It is the responsibility of each single interrupt server to de-assert its interrupt signal as part of its interrupt service routine.

Software Development

An interactive software development environment for uCore is rather straightforward and it has been realized under Linux and Windows on top of gforth under GNU GPL conditions. A "debuggable uCore" has an additional umbilical interface that can be controlled by a two-wire UART (RxD, TxD) interface on the host computer. The program memory, which must be realized as a RAM, can be loaded through this interface. After loading the application, a small debug monitor takes control and is exchanging messages with the host.

The following tasks can be done under control of a host computer:

- loading a program into uCore's program memory;
- resetting uCore;
- single-step debugging uCore with breakpoints;
- observing variables, buffer areas, semaphores, and the task list.

It loads on top of gforth (Windows and Linux), an open source 32 bit implementation of Forth. It can produce a binary image and a symbol table file

for the debugger, a VHDL file for simulation, and a MEM file for FPGA blockRAM configuration. Because the Forth systems are 32-bit systems, the cross-compiler only supports numbers of up to 32-bits signed magnitude. For larger data path widths, the cross compiler has to be extended accordingly if larger numbers need to be compiled.

It is a short but rather complex piece of code. Several peep hole optimizations have been implemented and the cross-compiler is of production quality.

lcc for MicroCore

We adapted the excellent ANSI-scc[2] [8] for generating MicroCore code. The scc is based on the lcc[3]. We highly recommend our few readers to get their hands on a copy of Hanson and Fraser's book [3] that describes in greater details the workings of lcc. We will here describe the interface between scc and Microcore in some detail, and only briefly skim (as need arises!) over the inner workings of the compiler.

Stack-based CPUs do not permit to address in a simple manner the single slots of the stack where the parameters for a procedure call are temporarily stored. All allocation algorithms that work well with register-based CPUs are useless for stack-based CPUs, since registers are individual random addressable slots of memory. The problem arises as how to allocate efficiently stack memory's slots for inter- and intra-procedure calls. Koopman [4] shows how to optimize the former, and Shannon [8] how to optimize the latter. Both algorithms are implemented in scc. For a good overview of the problem of global stack allocation, see [2].

The lcc consists of a front end and a back end. The former produces a meta representation of a C-program and the latter describes the ISA. The meta representation of the C-program is unfortunately not totally independent from the back end. A special structure (the interface) is responsible for the exchange of information between the two. We need to specify for example the data's alignment and type, the byte order (little or big endian), and for a typical lcc, the set of registers. In our case we must find a way to accommodate the stack structure of MicroCore within this register structure.

We will now discuss the changes we made in scc, to adjust it to the ISA of MicroCore.

The Meta representation

The C-Code is translated into a sequence of *Directed Acyclic Graphs* (DAGs) in such a way that each C-function is represented by a forest of DAGs. The DAG's nodes represent the primitive operations. Figure 6 shows the DAG-tree for the simple C-statement y = x[2] + 4. The processing of this

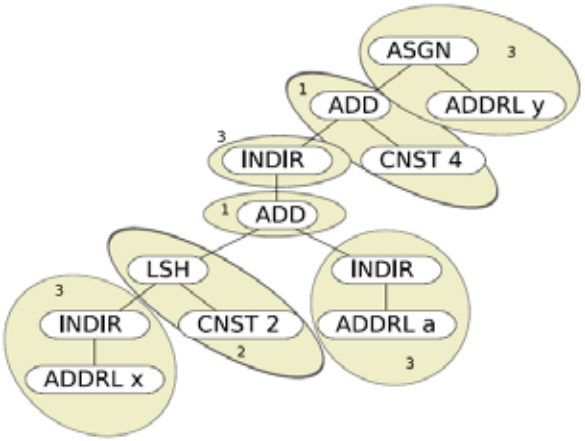

Figure 6: A typical DAG tree for the C statement: y = x[2] + 4. Picture adapted from [8].

DAG-tree necessitates following rules (for more details, see [6]):

```
stmt:   ASGNI4(stk, lAddr)     "#\n"       2
stk:    INDIRI4(lAddr)         "#\n"       2
stk:    CNSTI4                 "%a\n"      1
stk:    LSHI4(stk, stk)        "shift\n"  1+
                              (DATA_WIDTH+1)/2
```

DAG's nodes are assembled with three criteria: (1) type of operation, (2) data type and (3) size of data type. The following example illustrates the principle:

```
ADD + I + 4  ->  ADDI4
```

The size is always given in number of bytes. In our example the size is 4 bytes. Each node may have a variable number (between 0 and 2) of kids. Kids are nodes or terminal leaves. The interested reader will find in [6] the complete table with all types of nodes lcc supports.

We have defined following specific non-terminals for MicroCore:

- const: a constant;
- zero: constant 0;
- const1: constant 1;
- lAddr: the address of a local variable;
- addr: the address of a global variable, of a branch or of a dereferenced pointer (that means that all pointers contain absolute addresses.);
- jAddr: the address of a function;
- stk: represents a stack slot.

The full set of MicroCore's specific rules are shown in [6]. Here we illustrate them with a few examples. From the fact that all MicroCore's pointers may contain addresses or data, we derive the following rules:

```
addr:   INDIRP4(sread)     "\n"        0
addr:   INDIRP4(lAddr)     "#\n"       2
addr:   INDIRP4(addr)      "@\n"       1
```

2 scc = stack C compiler compiler
3 lcc = lean C compiler

Where:

```
lAddr: ADDRLP4 "%a"          1
/* No \n so must be printed by assig/indir */
addr:  CNSTP4 "#\n"          1
sread: COPYP "dup\n"         range(a, 1, 1)+1
sread: COPYP "over\n"        range(a, 2, 2)+1
sread: COPYP "2over\nnip\n" range(a, 3, 3)+2
sread: COPYP "any\n"         range(a, 0, 0)+1
```

For example, *addr: INDIRP4(addr) "@\n"* fetches data from a constant address of variable size. When scc sees the token '\#' in a rule, then it stops the evaluation of all the sub-trees which grow from that node, and it jumps to the function *emit2()* that calculates the exact size of the address (see [6]). The fourth column contains the cost (correct or guessed clock cycles per instruction) of the DAG-node. The compiler tries to construct a DAG-tree with minimal cost.

Optimizations

As we already observed, a stack-based CPU does not have single named registers that we can access in a random manner. On the contrary, we can access the slots of a stack only through stack operators like *swap*, *over*, etc. This fact makes all register allocation algorithms useless. We will not discuss here the equivalent algorithms for stack allocation, as they are very well explained in [4] and [2]. We give below a non-trivial example, which shows the working of these algorithms by generating MicroCore Forth code.

We examine the program mutual recursion to show how the intra- and inter-procedure optimization from [4] and [8] works.

```
int mRecFun(int cU, int cD){
    if(cD == 0){
        return cU;
    } else if(cD%2 == 0){
        return mRecFun2(++cU,--cD);
    } else {
        return mRecFun(++cU,--cD);
    }
}
int mRecFun2(int cU, int cD){
    if(cD == 0){
        return cU;
    } else if(cD%2 == 1){
        return mRecFun(++cU,--cD);
    } else {
        return mRecFun2(++cU,--cD);
    }
}
```

First we give in Listing 1 the MicroCore code without the local/global stack optimizations. The

```
( MicroCore(TM) uForth ) decimal : _mRecFun
    0 >r rsp@ 3 - rsp!
      ( allocate 3 local variables )
    6 l@  ?GOTO LABEL_2
        ( jump if not equal )
    5 l@ ( return int )
    GOTO LABEL_1
Label LABEL_2
    6 l@ 2 mod  ?GOTO LABEL_4
        ( jump if not equal )
    6 l@ 1- 3 l!
    3 l@ 6 l!
    3 l@ >r ( push int parameter 1 )
    6 l@ 1+ 3 l!
    3 l@ 6 l!
    3 l@ >r ( push int parameter 2 )
    CALL _mRecFun2  ( call to function )
    rdrop rdrop ( deallocate 2 parameters )
    ( tos ) 1 l!
    1 l@ ( return int )
    GOTO LABEL_1
Label LABEL_4
    6 l@ 1- 3 l!
    3 l@ 6 l!
    3 l@ >r ( push int parameter 1 )
    6 l@ 1+ 3 l!
    3 l@ 6 l!
    3 l@ >r ( push int parameter 2 )
    RECURSE ( recursive call to function )
    rdrop rdrop ( deallocate 2 parameters )
    ( tos ) 1 l!
    1 l@ ( return int )
Label LABEL_1
    rdrop rdrop rdrop rdrop
        ( deallocate 3 local Variables )
;
```

Listing 1: MicroCore code without stack optimizations

```
decimal : _mRecFun ( 2 parameters )
    swap swap dup 0 -
    ?GOTO lbl_2 ( jump if not equal )
    drop ( return int )
    GOTO lbl_1
  Label lbl_2
    dup 2 mod 0 -
    ?GOTO lbl_4 ( jump if not equal )
    swap 1+ swap swap swap 1-
    swap swap
    CALL _mRecFun2 ( call to function )
    ( return int )
    GOTO lbl_1
  Label lbl_4
    swap 1+ swap swap swap 1-
    swap swap
    RECURSE ( recursive call to function )
    ( return int )
  Label lbl_1
;
: _mRecFun2 ( 2 parameters )
    swap swap dup 0 -
    ?GOTO lbl_7 ( jump if not equal )
    drop ( return int )
    GOTO lbl_6
  Label lbl_7
    dup 2 mod 1 -
    ?GOTO lbl_9 ( jump if not equal )
    swap 1+ swap swap swap 1-
    swap swap
    _mRecFun ( return int )
    GOTO lbl_6
  Label lbl_9
    swap 1+ swap swap swap 1-
    swap swap
    RECURSE ( recursive call to function )
    ( return int )
  Label lbl_6
;
```

Listing 2: MicroCore code with stack optimizations

multiple use of l@ means that the local variables are temporarily stored outside the stack, thus increasing the time to access them. The optimized version of the program is given in Listing 2.
All the temporarily defined local variables remain on the stack and are accessed through normal stack operations like -*rot*, *dup* or *drop*.

The peephole optimization is carried out by the Forth cross-compiler whose responsibility is the transfer of the MicroCore Forth code to the target.

The cryptological function in Hardware

We used the BLAKE hash algorithm [1] in order to locate those cryptological functions that consume the most CPU cycles. The whys of this choice are explained elsewhere [5][6]. BLAKE security was evaluated by NIST in the SHA-3 process as having a "very large security margin", and the cryptanalysis published on BLAKE was noted as having "a great deal of depth". The BLAKE hash function comes in a family of four hash functions: BLAKE-224, BLAKE-256, BLAKE-384 and BLAKE-512. BLAKE-256 is based on 32-bit words, whereas BLAKE-512 on 64-bit words, from which the other members of hash functions are derived.

BLAKE uses only three basic operations: integer addition, xor '⊕', and rotation '<<<' (i.e., a circular shift given by a prescribed shift amount).

The G-function is the core of BLAKE and the source of its security against differential attacks [1]. Each G-function of BLAKE, $G_i(a,b,c,d)$ at round r operates on four selected words a,b,c,d of a 4 x 4 square of state words, and sets:

$$
\begin{aligned}
a &:= a+b+(m_{\sigma(r,2i)} \oplus c_{\sigma(r,2i+1)}) \\
d &:= (d \oplus a) \lll 16 \\
c &:= c+d \\
b &:= (b \oplus c) \lll 12 \\
a &:= a+b+(m_{\sigma(r,2i+1)} \oplus c_{\sigma(r,2i)}) \\
d &:= (d \oplus a) \lll 8 \\
c &:= c+d \\
b &:= (b \oplus c) \lll 7
\end{aligned}
$$

Here, σ denotes a specified permutation of numbers between 0 and 15, and the m_k denotes message words. The G-function performs six integer additions, six xors, and four word rotations. Apart from the permutation σ, the time consumption of each of the operations involved in the computation of a G-function has to be inspected. The rotation amounts in the second and fourth assignments differ, and are not always a power of two.

Operation	Seconds for 10^8 operations
Integer addition	0.18
xor	0.18
rotation	0.20

Table 6: Experimental results for BLAKE basic operations

Performance with gcc

For performance measurements, a computer with the following technical characteristics has been used: CPU Intel(R) Core i7 M 640 at 2.80 GHz; 2 GiB DIMM-Memory at 1333 MHz, 4 MiB external Cache, 320 GiB of permanent storage.

As the G-function is composed of the operations, (unsigned) integer addition, xor and rotation, the time consumption of these operations is measured and compared individually. To obtain reproducible and measurable quantities, each measurement has been executed 10^8 times (see Table 6). We give in Listing 3 the reference implementation of the G-function.

The measurements have been done using the C-function *clock()*. Timings for differing rotation amounts in the rotation operation don't vary significantly. Note however, that rotation amounts that are a multiple of 8 can be implemented by just reordering bytes, which is often faster than shifting the words. For measurements, rotation has been implemented as in the official BLAKE document, as an ANSI C-preprocessor macro. As a result, rotation is the most time consuming operation (see Table 6). When looking at assignments, the first and fifth assignments are more time consuming than the others, as they involve more basic operations plus a permutation of 16 words.

Performance with scc

We only show an example of the performance measurements before and after the hardware implementation of the *ROT32(x,n)* function. The full set of measurements is available upon request. We

```
#define ROT32(x,n) ((((x) << (32-(n))) | ((x) >> (n))))
#define ADD32(x,y) (((u32)((x) + (y))))
#define XOR32(x,y) (((u32)((x) ^ (y))))

#define G32(a,b,c,d,i) \
  v[a] = (ADD32(v[a],v[b]) + XOR32(m[sigma[round][2*i]], c32[sigma[round][2*i+1]])); \
  v[d] = ROT32(XOR32(v[d],v[a]),16); \
  v[c] = ADD32(v[c],v[d]); \
  v[b] = ROT32(XOR32(v[b],v[c]),12); \
  v[a] = (ADD32(v[a],v[b]) + XOR32(m[sigma[round][2*i+1]], c32[sigma[round][2*i]])); \
  v[d] = ROT32(XOR32(v[d],v[a]), 8); \
  v[c] = ADD32(v[c],v[d]);
```

Listing 3: Reference implementation of the G-function

```
Literal:        Bit 7=1           Opcode:         Bit 7=0

Forth Stack Description

Type     Stack     Instr.      LIT     Before     After     Notes      Number of Cycles (CPI)

BRA: Branch Functions (32 possibilities: 32 used)
...
BRA      BOTH      ROT32       *       n          --        ROT32      1
...
```

Listing 4: Function ROT32 defined as new Micro-Forth word

measured one full round of the loop at the heart of BLAKE containing all the $G_i(a,b,c,d)$ functions:

```
...
/*  do 12 rounds */
for(round=0; round < NB_ROUNDS32; ++round) {
    _ledsOn_();

    /* column step */
    G32(0, 4,  8, 12, 0);
    G32(1, 5,  9, 13, 1);
    G32(2, 6, 10, 14, 2);
    G32(3, 7, 11, 15, 3);

    /* diagonal step */
    G32(0, 5, 10, 15, 4);
    G32(1, 6, 11, 12, 5);
    G32(2, 7,  8, 13, 6);
    G32(3, 4,  9, 14, 7);

    _ledsOff_();
}
...
```

All times are measured with a Tektronix TDS 2014 oscilloscope with a resolution of 2 GHz directly on the MicroCore board (see Fig. 4). 12 rounds without hardware optimization take 2876 µs. With hardware optimization they only take 1700 µs, which results in a speedup of 1.69. The experiments show that the choice of parameters for $G_i(a,b,c,d)$ does not influence very much its performance. Namely, the average duration of a $G_i(a,b,c,d)$ function within the loop is: 1700/12/8 = 17.7 µs.

Discussion

Our measurements confirm that the apparently trivial function *ROT32(x,n)* is the performance bottleneck of BLAKE. It is interesting to note that this function is used very often in different hash-functions. Therefore, we decided to implement it in VHDL and to define it as new Micro-Forth word (and as part of the MicroCore ISA) as given in Listing 4.

It is quite remarkable that this new instruction has a CPI of 1. We have thus demonstrated that MicroCore is a very elegant platform that permits a continuous transition between hardware and software optimization.

Acknowledgement

We thank Dr. Crispin Bayley of York University for the source code of the global allocation algorithms.

References

[1] Jean-Philippe Aumasson, Luca Henzen, Willi Meier, and Raphael C.-W. Phan, SHA-3 proposal BLAKE, http://www.131002.net/blake/, 2010. 14, 15.

[2] Chris Bayley and Mark Shannon, Register allocation for stack machines, Proceedings of the 22nd EuroForth Conference (2006), 13–20. 10, 12.

[3] Christopher W. Fraser and David R. Hanson, A retargetable C compiler: design and implementation, Addison-Wesley, 1995. 10.

[4] Philip Koopman Jr., A preliminary exploration of optimized stack code generation, Proceedings of the rochester Forth Conference (1992), –. 10, 12, 13.

[5] Markus Knecht, Willi Meier, and Carlo U. Nicola, A space- and time-efficient implementation of the Merkle tree traversal algorithm, http://arxiv.org/abs/1409. 4081, 2014. 14.

[6] Carlo U. Nicola, Markus Knecht, Willi Meier, and Klaus Schleisiek, Microcore: An optimized architecture for cryptological functions, Tech. report for the Hasler Foundation, FHNW, October 2014. 11, 12, 14.

[7] Klaus Schleisiek, MicroCore 1.66 open, scalable, dual stack, Harvard processor for embedded control, June 2011. 2.

[8] Mark Shannon, A c compiler for stack machines, Master's thesis, University of York, Department of Computer Science, 2006. 10, 11, 13.

Fachhochschule Nordwestschweiz
Institut für Mobile und Verteilte Systeme
Bahnhofstrasse 6
CH-5210 Windisch

www.fhnw.ch/technik/imvs
Tel. +41 56 202 99 33