

# configKIT - A Security-Aware Software-Configuration-Tool for Wireless Sensor Networks

## Demo Submission

Steffen Peter , Krzysztof Piotrowski and Peter Langendörfer  
IHP GmbH, Im Technologiepark 25, 15236 Frankfurt (Oder), Germany  
{peter,piotrowski,langendoerfer}@ihp-microelectronics.com

### Keywords

Sensor networks, software configuration, security

## 1 Introduction

The demo presents a security centric configuration tool – configKIT – for software running on wireless sensor nodes. It selects modules from a library and combines them to a valid system satisfying application requirements given by the developer. In the selection process configKIT respects semantic and syntactic requirements of software and hardware, resolves dependencies, estimates memory consumption, and classifies the energy consumption. Additionally it emphasizes the classification of security properties, i.e. secrecy, robustness, or integrity, of the explored configurations. This function assist also non-security-experts in making their WSN-implementation more secure. The demo allows users to generate fine tuned solution even for rather general formulated application and security requirements, and to experience how even slight modifications of requirements can significantly change the recommended software configuration.

### Background

Highly specialized and optimized modules are a promising approach to cope with the severely constraint resources of wireless sensor nodes. The direct connection of these small modules without using a complicated structure or inter-module communication appears to be the most efficient solution and is widely accepted. The idea has been realized for example in TinyOS, which merges all required components to one block, which is memory efficient and results in reduced computational overhead. Beside the problem of keeping an overview of a large set of modules, the major issue in this approach is development and maintenance costs. The substitution of one module may affect many other components. This is particularly the case with respect to security properties. An alternation of a security profile that eventu-

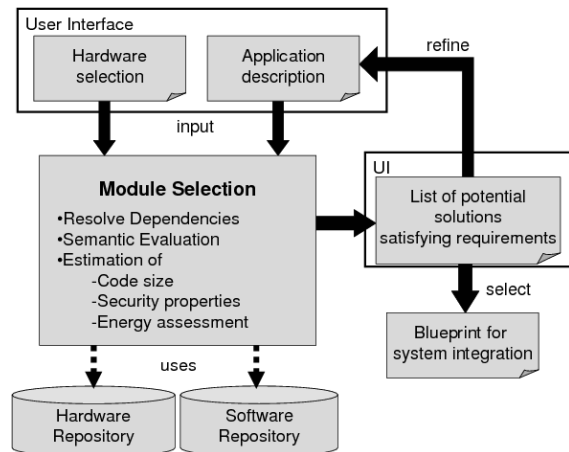


Figure 1. Flow of configKIT's selection process

ally exchanges several security protocols and modules would correspond to a completely new development of the system. The demo shows how configKIT copes with the enormous set of state-of-the-art approaches and implementations and still provides good accessibility.

## 2 Demonstrated Research Contributions

The general notion of configKIT's selection process is shown as Figure 1. Based on an application- and hardware selection and using repositories describing the properties and footprints of hardware and software components, a module selection algorithm computes configurations that:

- can be combined to a single software image running on the intended node (compatibility of interfaces)
- are compatible to the available hardware (compatibility to hardware)
- fulfill the semantic and functional requirements of the applications
- fulfill the security requirements

The resulting software configuration contains suitable modules and a prediction of the eventual memory and energy consumptions, as well as an assessment of the security properties, even for combined security features such as data integrity. The tool presents the proposed software configurations, which can be further refined or finally selected for system integration. The result is a new and unique composition of modules in each configuration process. By that the

#### APPLICATION NEEDS:

**HARDWARE**  
MicaZ

**SOFTWARE**  
Event Based  
Signature  
DSDV

Integrity:	<input type="radio"/> 0 <input type="radio"/> 0 <input type="radio"/> 1 <input type="radio"/> 2 <input type="radio"/> 3
Secrecy:	<input type="radio"/> 0 <input type="radio"/> 0 <input type="radio"/> 1 <input type="radio"/> 2 <input type="radio"/> 3
Robustness:	<input type="radio"/> 0 <input type="radio"/> 0 <input type="radio"/> 1 <input checked="" type="radio"/> 2 <input type="radio"/> 3
Energy:	<input type="radio"/> 0 <input type="radio"/> 0 <input type="radio"/> 1 <input type="radio"/> 2 <input type="radio"/> 3

Integrity:	<input type="radio"/> 0 <input type="radio"/> 0 <input type="radio"/> 1 <input type="radio"/> 2 <input checked="" type="radio"/> 3
Secrecy:	<input type="radio"/> 0 <input type="radio"/> 0 <input type="radio"/> 1 <input type="radio"/> 2 <input type="radio"/> 3
Robustness:	<input type="radio"/> 0 <input type="radio"/> 0 <input type="radio"/> 1 <input type="radio"/> 2 <input type="radio"/> 3
Energy:	<input type="radio"/> 0 <input type="radio"/> 0 <input type="radio"/> 1 <input type="radio"/> 2 <input type="radio"/> 3

Integrity:	<input type="radio"/> 0 <input type="radio"/> 0 <input type="radio"/> 1 <input type="radio"/> 2 <input type="radio"/> 3
Secrecy:	<input type="radio"/> 0 <input type="radio"/> 0 <input type="radio"/> 1 <input type="radio"/> 2 <input type="radio"/> 3
Robustness:	<input type="radio"/> 0 <input type="radio"/> 0 <input type="radio"/> 1 <input type="radio"/> 2 <input type="radio"/> 3
Energy:	<input type="radio"/> 0 <input type="radio"/> 0 <input type="radio"/> 1 <input type="radio"/> 2 <input type="radio"/> 3

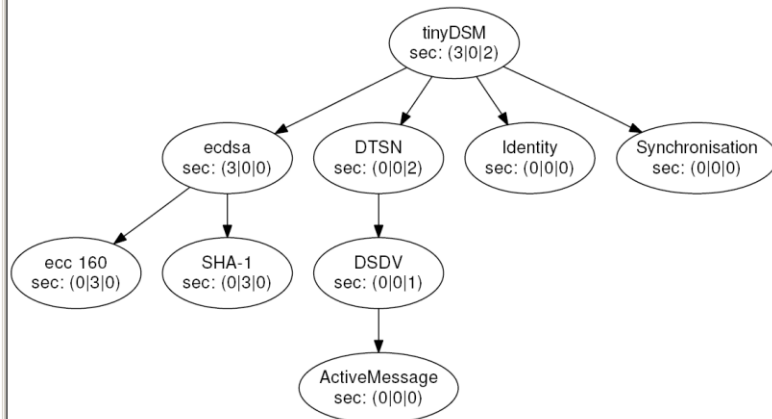


Figure 2. Partial screen shot of the selection dialog on the left and a resulting dependency graph on the right

configKIT approach significantly improves the state-of-the-art normally consisting of prototypical application frameworks that can be slightly tailored for individual purposes. Since configKIT delivers complete software configurations early in the development process and extremely fast (i.e. in less than a few seconds), it can help avoiding design errors and therefore can be an important step in the development of WSNs.

The configKIT demonstration presents a set of novel contributions:

**The configKIT-supported development flow** includes setup and exploration phase and helps coping with the complexity of the WSN-design-process. It is outlined in the previous paragraph.

**The security assessment approach** functionally combines security properties of each module from the the bottom to the top of the system. On the paths through the dependency graph toward the top module the properties are derived and transformed. For instance a signature algorithm transforms secrecy of a hash and an asymmetric cipher to an integrity property.

**A functional classification of the modules** allows to address sets of modules in an intuitive way, and is key for the definition of functional application requirements.

**Heuristics for memory and energy consumption** help to estimate and compare the system's footprint early in development process. It is a classification these properties that allows comparison in the selection process.

**The selection algorithm** is initially a rather straightforward branch-and-bound algorithm with optimizations, sufficiently boiling down the practical complexity.

**Semantic and syntax of the description languages** are XML-based, open and expendable.