ABSTRACT
This poster presents a novel memory management scheme for small form factor secure microcontrollers, notably those embedded in smartcards. The scheme is currently being deployed as a part of the Code Slicing Operating System (COSMOS), a novel operating system that extends the security perimeter of a smartcard in order to enable complex multi-applications. We present how the memory management scheme abstracts memory using virtualization and how the virtualization of memory enables an application to be run in many different hardware settings, with different (particularly memory) constraints. In addition, the proposed scheme provides many security features that are required by applications that use secure microcontrollers. In particular, it supports isolation between applications and a flexible access control of memory to only those authorized to do so. We demonstrate that the overhead added to code execution by the technique is small by implementing the memory management scheme in a secure microcontroller. Even though the present scheme is designed within a specific OS, many of the presented approaches can be used in other environments that require memory to be virtualized, particularly in resource constrained environments.

PERFORMANCE EVALUATION
The main overheads created by the technique are:
• Clock Cycles Overhead

<table>
<thead>
<tr>
<th>Instance</th>
<th>Clock Cycles</th>
</tr>
</thead>
<tbody>
<tr>
<td>C-JNE</td>
<td>23</td>
</tr>
<tr>
<td>C-JNE</td>
<td>23</td>
</tr>
<tr>
<td>BGE</td>
<td>32</td>
</tr>
<tr>
<td>ADD</td>
<td>32</td>
</tr>
</tbody>
</table>

2. Slice Download Overhead
3. Slice Resolution Overhead

SLICING MANAGER
Each memory access during a program’s execution can be thought of as a model as a reference to a slice. This reference has to be translated into a physical memory address. This way, the reference has to be interpreted by a central memory manager that will return the correct physical address of the referenced slice. This memory manager has to check whether the referenced slice is currently loaded on the card. If the slice is not inside the card, the memory manager has to download the slice, decrypt it, and verify its authenticity. After making sure the referenced slice is inside the card, the memory manager will proceed with returning the proper physical address. We define this central memory manager as part of the Slicing Manager (SLMG). This way, the SLMG has a Virtual Machine sub-module that is responsible for replacing intermediate representation assembly instructions by machine code that performs the following:
1. It checks if calling the SLMG is necessary (different slice) and redirects execution flow if it is the case.
2. It carries out the original intended assembly instruction with the actual physical address(es) returned by the SLMG.

The intermediate representation assembly code will be generated by the compiler. Memory in this representation will be composed by a slice tag, which uniquely identifies a slice, and an offset in that slice. The compiler, which is also in charge of generating slices, will refer to any memory address as this combination of slice tag and offset. The optimal numbers of bits allocated to the slice tag and to the offset are still being investigated. The resulting intermediate representation assembly code will be interpreted by the SLMG’s virtual machine in the form of code replacement.

Absolute JMP Pseudo Code
Original:
```
jmp #ADDR32;
```
Converts into:
```
Address:
#ADDR32 =
#14bits_base_high, #14bits_base_low, #12bits_offset

Code:
```
//If Address is in a different slice/jump to last instruction.
cjne sltag_high, #14bits_base_high, +2

//If offset exceeds the slice limit branch to seg_fault.
bge slsize, #10bits_offset, seg_fault:

//Sum offset + slice’s base physical address and store the
//result in Instruction Pointer, thus branching to the
//proper 16 bits address.
add IP, sl_padd, #10bits_offset

//Branch to related SLMG’s function if address in a
different slice so the proper slice is made active.
jmp sm_jmp:

The following assembly pseudo code explains how the SLMG accomplishes the task of maintaining state information while executing the aforementioned absolute jmp instruction.
```
sm_jmp:
1. Save Stack and Registers
2. If slice is Outside (not in slices table) then:
3. Load slice onto the card (VIO)
4. Decrypt the slice
5. Check Slice’s signature
6. Update SLMG state (slice table)
7. Update SLMG state (sltag_low, sltag_high, sl_padd, slsize)
8. Calculate 16 bits physical address from 32 bits virtual address, checking slice boundary
9. Restore stack and registers
10. Update Instruction Pointer with target 16 bits address (actual jmp)

As we can see, the SLMG’s function sm_jmp handles all the process of slice resolution whenever an assembly instruction reference a slice different from the one that is running. The SLMG has different functions for different assembly instructions, but all of them have the same inner structure. The address calculation in sm_jmp’s step 8 is performed similarly to what is done in pseudo code 2. The routine simply adds the 10-bit offset to the current slice base physical address (the physical address that points to the beginning of the slice). After restoring stack registers, the result sum is stored in the Instruction Pointer (IP), thus simulating an actual jmp instruction

REFERENCES
2. Davis, Patty et al.; “U.S. spy plane, Chinese fighter collide”; CNN, April 1, 2001
6. Davis, Patty et al.; “U.S. spy plane, Chinese fighter collide”; CNN, April 1, 2001