SCuP - Secure Cryptographic Microprocessor

Roberto Gallo[12], **Henrique Kawakami**[1], **Ricardo Dahab**[2*]

[1]KRYPTUS Security Solutions Ltd., Campinas, SP, Brazil

[2]Campinas State University, Campinas, SP, Brazil

{gallo,rdahab}@ic.unicamp.br, {gallo,kawakami}@kryptus.com

Abstract. *In this paper we present SCuP - the Secure Cryptographic Micro-Processor with secure code execution (encrypted, signed). SCuP is an asymmetric multicore processor for general applications with several innovative protection mechanisms against logical and physical attacks. Among the main processor features are the hardware firewall (HWF) and the deep inspection/introspection mechanism (MIP) along with the secure execution packages (PES). SCuP has been validated in simulations and in FPGAs and its semiconductor diffusion will be done in the next few months.*

Resumo. *Neste artigo apresentamos o SCuP - Processador Criptográfico com Execução Segura de Código (cifrada, assinada). O SCuP é um processador de múltiplos núcleos assimétrico para aplicações gerais, que apresenta diversos mecanismos inovadores de proteção contra ataques lógicos e físicos ao processador. Dentre as principais características do processador estão o firewall de hardware (HWF) e o mecanismo de inspeção/introspecção profunda (MIP) combinados com os pacotes de execução seguros (PES). O SCuP foi validado em simulações e em FPGAs e deverá seguir para difusão semicondutora nos próximos meses.*

1. Introdução

A importância da segurança nos sistemas baseados em hardware e software é bem estabelecida e dispensa justificativas. Entretanto, apesar de sua relevância, sistemas computacionais seguros têm se mostrado supreendentemente difíceis de serem obtidos. Parte desta dificuldade pode ser atribuída ao fato de que segurança mais do que uma área do conhecimento, é uma transversal que perpassa diversas áreas, como Teoria do Números, Codificação Segura, Criptografia, Estatística e Física dentre outras. Desta forma, até o momento não existe uma teoria unificada para Segurança, o que explica as recorrentes falhas reportadas em toda sorte de sistemas.

O SCuP foi desenvolvido dentro da visão mais ampla de segurança e que considera que quaisquer componentes dos sistemas podem conter defeitos de segurança.

Nossa Contribuição Neste artigo apresentamos o SCuP - o Secure Cryptographic Microprocessor, um processador de uso geral cuja arquitetura foi projetada para garantir altos níveis de proteção e resiliência mesmo contra os adversários mais motivados com um cenário de ataques ampliado. Entre as características que tornam o SCuP único estão: i) o emprego de múltiplos núcleos com processamento assimétrico; ii) mecanismos de

inspeção e introspeção de software; iii) suporte a mecanismos de reparação dinâmica de software; e iv) mecanismos de execução segura de pacotes.

Organização do Artigo Na Seção 2 fazemos uma ampla revisão de problemas e soluções de sistemas seguros; na Seção 3 apresentamos a arquitetura do SCuP e os seus componentes; na Seção 4 apresentamos alguns resultados de implementação; a Seção 5 conclui e apresenta algumas possibilidades de trabalhos futuros.

2. O Problema e Trabalhos Relacionados

Processadores e sistemas seguros estão relacionado com, entre outras, as áreas de: i) arquiteturas seguras de hardware; ii) co-processadores seguros; iii) prevenção, detecção e recuperação de violação de segurança; iv) métricas de segurança; e v) interfaces seguras.

Co-Processadores Criptográficos

Os trabalhos relacionados mais relevantes, que abrangem mais de uma área mencionada, incluem o trabalho pioneiro do desenvolvimento do módulo criptográfico IBM4758 [4], precursor de diversos dos mecanismos de segurança atualmente empregados em hardware seguro, principalmente do ponto de vista de segurança física. O IBM4758 é um dispositivo (placa) PCI, multi-chip, com funções de Hardware Security Module (HSM) e também capaz de executar programas de usuários, previamente assinados, em seu processador com arquitetura 80486.

Apesar de seu elevado nível de segurança física, o IBM4758 é inapropriado para diversos cenários de uso. Neste sentido, a arquitetura AEGIS [20] representa uma evolução importante ao propor um processador (em um único componente) capaz de realizar a execução segura de código utilizando os conceitos de cadeia de confiança (que parte de um processo de boot seguro) e o isolamento de processos seguros daqueles não seguros por meio de modificações de arquitetura em um núcleo de processador MIPS . O AEGIS também emprega de maneira inovadora a proteção da memória RAM (off-chip) do processador por meio da cifração e autenticação do conteúdo de memória.

O processador Cerium [2] é uma outra proposta também relevante, menos completa do ponto de vista de arquitetura, mas que traz um benefício importante: saídas certificadas (assinadas) da execução de aplicações. Com isso, entidades externas (clientes ou atestadores) podem confiar nos resultados da computação através da verificação das assinaturas produzidas. Há uma clara diferença de visão em relação aos trabalhos anteriores: o interessado na integridade de um sistema não necessariamente é o seu proprietário, a exemplo de aplicações de DRM (Digital Rights Management).

Execução Segura de Código

As aplicações de DRM estão sujeitas a um modelo de ameaças (threat model) particularmente interessante: se por um lado conteúdo (aplicações, músicas, filmes) pode custar milhões de dólares para ser produzido, o ganho do adversário individual com a pirataria do mesmo conteúdo é ordens de grandeza menor; ou, de outra forma, a motivação de um

adversário para copiar alguns arquivos de música é muito limitada, em especial se o custo do "ataque" for proporcional ao número de arquivos copiados.

Esse modelo de ameaças foi aquele utilizado na concepção da geração atual do padrão do Trusted Platform Module (TPM) do Trusted Computing Group (TCG) [22], a plataforma (hardware + software) padrão de mercado para computação confiável em dispositivos como computadores pessoais e aparelhos celulares. O TPM-TCG é um periférico soldado à placa mãe do sistema e que possui capacidades de assinatura digital, verificação de assinatura e software *measurement*. Em um sistema habilitado, o TPM pode ser utilizado para a verificação da cadeia de boot do sistema e, após inicializado, na verificação (*measurement*) da integridade das aplicações em execução.

A proposta do TPM tem alguns méritos: i) tem baixo custo; ii) não requer refatoração de código legado; e iii) vai no caminho certo ao considerar tanto integridade de binários como de imagens em execução. Por outro lado, possui sérias limitações: i) é um dispositivo escravo de barramento, podendo ser completamente ignorado pelo sistema, e também não possui poder de inspeção; ii) possui arquitetura similar a de um smartcard, com barramento LPC, resultando em baixa banda de comunicação com sistema e baixo poder computacional; iii) pode ser subvertido por meio de modificações na BIOS do sistema (na sessão CRTM); e iv) não considera sigilo, relegando às aplicações essa tarefa.

De forma a melhorar o perfil de segurança sem aumento considerável de custos, Costan et al [3] propuseram e implementaram (utilizando um processador Javacard) o Trusted Execution Module (TEM), capaz de executar código seguro no próprio módulo, através de Secure Execution Closure Packs (SECpacks). Os SECpacks permitem que aplicações de tamanho arbitrário, especialmente escritas, sejam fatoradas em pequenos módulos e executadas no ambiente embarcado seguro (smartcards, processadores seguros) ao custo de operações de E/S adicionais e da degradação de performance que acompanha a fatoração.

O emprego de pacotes de execução segura no TEM remonta, possivelmente, ao IBM4758, mas foi no IBM Cell [19], utilizado no Sony Playstation 3, que ela foi utilizada de uma forma mais consistente. O Cell é um processador multi-núcleo assimétrico especialmente voltado para o mercado de entretenimento, onde poder de processamento e proteção de conteúdo têm prioridades altas. De especial interesse no Cell são os Synergistic Processing Elements (SPE), responsáveis pelo processamento de alto desempenho do processador. Cada SPE pode ser colocado em modo de execução seguro (com código e dados assinados e cifrados), isolado dos demais núcleos, no modo *secure processing vault* - com memória interna própria. Neste modo nenhum outro núcleo é capaz de inspecionar ou alterar código ou dados em execução. Os ganhos são claros: aumento do nível de proteção contra pirataria ao diminuir o risco de que fragilidades nos softwares executando nos demais núcleos sejam utilizadas para atacar o SPE no modo *vault*.

A execução segura de pacotes pode ser vista como um tipo especial de isolamento de threads, ou execução segura de threads, onde o número de processos executando simultaneamente no processador é limitado ao número de núcleos; ou, de outra forma, threads seguras não coexistem com threads normais em um mesmo núcleo.

A execução de threads seguras (ou isoladas) simultaneamente em um mesmo núcleo foi implementada tanto na proposta do AEGIS por meio de instruções e modos

de execução privilegiados, como mais recentemente na arquitetura SP [9, 13]. A arquitetura SP, no entanto é de uso mais geral e minimalista e pode ser aplicada com menor número de intervenções em arquiteturas de processadores já existentes, como as famílias x86 e SPARC. A Arquitetura SP utiliza alterações de *instruction sets* e a adição de componentes de cifração de memória; e, utilizando o proposto Trusted Software Module, um sistema operacional seguro minimalista, provê serviços de confidencialidade e atestação remotos.

Arquiteturas para Execução Monitorada

Apesar de não apontado pelo trabalho de Lee [9], podemos conjecturar que, com modificações no TSM, a arquitetura SP (e também na pilha de software do AEGIS) poderia ser utilizada para introspecção de software entre processos. Esse uso, no entanto, parte do princípio de que o sistema verificador (SV) não sofre das mesmas fragilidades que o sistema em verificação (SEV), e que também não é influenciado por alguma execução faltosa. Para diminuir riscos implícitos de segurança provenientes de problemas de implementação e arquiteturas de solução complexas, muito se fala [18, 9] da utilização de bases minimalistas de computação confiada onde a pilha de software (BIOS segura, S.O. seguro, aplicações seguras...) é reduzida a alguns poucos milhares de linhas de código.

Entretanto, tanto quanto saibamos, nenhum trabalho tem se atentando ao fato de que as arquiteturas de hardware de processadores (e sistemas) são descritas em centenas de milhares ou mesmo milhões de linhas de código de linguagens de descrição de hardware e, portanto, estão sujeitas a problemas de implementação tanto quanto os softwares, na medida em que segurança não é uma consideração comum no mundo dos sintetizadores de hardware. Desta forma, é temerário esperar que um sistema típico não possua problemas ocultos de segurança em termos de implementação.

Trabalhos como CuPIDS [23] e CoPilot [15], por sua vez, caminham por uma linha de pesquisa distinta: utilizam pares de sistemas, um monitor de (políticas) de segurança e outro monitorado. O CoPilot é voltado para o monitoramento e recuperação de ataques de rootkits. Ele é implementado por meio de uma placa PCI-mestre de barramento (sistema monitor), conectada a um hospedeiro (sistema monitorado) e é capaz de inspecionar todo o seu espaço de endereçamento.

O monitor não compartilha recursos com o sistema monitorado; assim, em caso de instalação de um rootkit no sistema principal, a placa PCI é capaz de verificar que houve modificações no espaço de endereçamento do kernel do sistema e assim corrigir o sistema e avisar uma estação de monitoramento externa. **Por possuírem arquiteturas completamente diferentes, monitor e monitorado também minimizam a possibilidade de compartilharem defeitos**.

Já as limitações do CoPilot estão principalmente ligadas à degradação de desempenho causada pelo processamento, pelo monitor, do espaço de endereçamento do sistema monitorado, o que restringe a usabilidade da proposta à verificação do kernel do sistema em RAM. O custo do hardware também é um problema.

No CuPIDS, por sua vez, a ideia de sistema independente de monitoramento é revisitada com uma nova arquitetura de hardware e novos objetivos de monitoramento.

Com o uso de uma *motherboard* com dois processadores, seus autores dividem o sistema em porção monitora e porção monitorada. Ao contrário do CoPilot, que tem como alvo o próprio sistema operacional, no CuPIDS existe, para cada serviço implementado na porção monitorada, um co-serviço de monitoramento na porção monitora (possivelmente por meio de uma política *EM-enforceable* [18]). Os potenciais problemas com o CuPIDS estão ligados à garantia da própria integridade da porção monitora. Sendo implementados em processadores de uso geral, estão sujeitos a diversos tipos de ataque, como substituição de binários, por exemplo.

Integridade dos Sistemas

Em aplicações onde a **integridade** dos **serviços prestados** pelo sistema (mais do que a integridade do próprio sistema) é preocupação primordial, diferentes técnicas têm sido utilizadas, em especial na área de sistemas de votação. Sistemas de votação eletrônica devem atingir simultaneamente objetivos aparentemente inconciliáveis: i) um voto por eleitor; ii) voto registrado conforme a intenção (do eleitor); iii) voto contado conforme o registro; iv) sigilo do voto; v) verificabilidade do voto; e vi) resistência à coerção [17]. Quaisquer tentativas de se atingir estes objetivos têm implicações diretas na concepção das máquinas de votação (digital recording electronic voting machine - DRE).

Admite-se, como regra, que os sistemas não são confiáveis e que podem ser adulterados. Desta forma, mecanismos eficientes de verificação da integridade do sistema e dos próprios serviços devem ser implementados e imediatamente acessíveis aos interessados. A integração entre integridade dos sistemas e os usuários foi explorada em trabalhos como VoteBox Nano [12], assim como em [5, 6], utilizando a noção de **caminhos confiados** e **classe de nível de confiança**.

A confiança obtida pela verificação do sistema em produção, no entanto, sempre está ligada (e limitada) pela confiança nas fases de desenvolvimento, ou no ciclo de vida, do dispositivo, como apontam Mirjalili e Lenstra [10]. Neste sentido, padrões como o FIPS 140-2 [11] e o Common Criteria [21] têm papeis relevantes. O padrão FIPS 140-2 apresenta um conjunto de requisitos e recomendações que um módulo criptográfico de uso específico (geração, guarda e uso de chaves criptográficas) deve obedecer. Apesar de não fixar diretamente nenhum item de arquitetura de tais módulos, o padrão é relevante por ser completo nos diversos aspectos de segurança que um módulo deve satisfazer (proteções lógicas, físicas, controle de acesso, sensoriamento, auto-testes, etc).

Verificação dos Sistemas e Padrões

O Common Criteria, por sua vez, é um meta-padrão, que define *templates* sobre os quais perfis de segurança os (*security profiles*) devem ser elaborados, e que descreve as características esperadas de um dado sistema (seguro), o qual é mais tarde certificado com base no próprio perfil. A principal contribuição do CC é a listagem ampla de itens que devem ser cobertos por um perfil de segurança.

No quesito verificação, de uma forma mais ampla, a nossa proposta está marginalmente relacionada aos trabalhos de verificação formal de sistemas, onde componentes

(lógicos) de software e hardware são descritos formalmente e técnicas de obtenção de provas são utilizadas para se determinar a validade das especificações. Apesar de poderosas, no entanto, tais técnicas têm complexidade NP-difícil ou indecidível [7, 8, 16], dificultando o seu uso na prática. Desta forma, técnicas totalmente informais ou mistas são utilizadas na verificação das propriedades dos sistemas [1]. Neste sentido, técnicas de verificação lógica de segurança baseadas em simulação, em especial via introspecção de máquinas virtuais, têm se mostrado úteis, como mostram os trabalhos de Payne [14] e Dwoskin [13].

3. Arquitetura do SCuP

A Figura 1 mostra a arquitetura do SCuP. Nela é possível identificar dois núcleos SPARC de 32 bits, baseados no Leon 3, instanciados de maneira assimétrica, o Application Core - AC, e o Secure Core - SC. Estes dois núcleos estão ligados aos barramentos internos AHB (e APB) modificados. Estes barramentos implementam um firewall de hardware, programado por SC e que limita o acesso dos mestres de barramento aos periféricos.

Os componentes periféricos encontrados no processador são divididos em dois principais grupos: componentes sem função de segurança (caixas mais claras e que incluem, dentre outros, USB, PCI, Controle de DDR2) e componentes com funcionalidades seguras (como cifradores de memória externa e o TRNG (*true random number generator*)). No que se segue apresentamos uma breve descrição dos componentes do SCuP e em seguida algumas das funcionalidades de segurança permitidas pela nossa plataforma.

3.1. Componentes do Sistema

3.1.1. Os Núcleos AC e SC

A arquitetura do SCuP permite que coexistam diversos (n) Núcleos de Aplicação (AC) com diversos (m) Núcleos de Segurança (SC). Na Figura 1, $n = m = 1$. O AC é um núcleo completo para uma CPU, com unidade de ponto flutuante, cache de dados e programa e MMU, e que serve para a execução de aplicações de usuários convencionais como, por exemplo, executar um S.O. Linux com uma aplicação de votação. O AC é controlado e monitorado pelo SC, com mecanismos que serão descritos mais a adiante.

O SC, por sua vez, é um núcleo minimalista e que foi modificado para ser uma das raízes de confiança do sistema. Para minimizar possibilidades de defeitos de segurança no próprio VHDL do processador, FPU e MMU foram eliminados, ao mesmo tempo em que uma memória interna, de acesso exclusivo ao núcleo foi adicionada. Esta memória, chamada de scratchpad, é essencial na segurança do sistema e foi projetada para permitir que operações que demandam sigilo (manipulação de chaves e outros parâmetros críticos de sistema) pudessem ser realizados com a menor possibilidade de vazamento e sem a necessidade de memória externa. O SC tem capacidade para executar um sistema operacional mínimo, seguindo o princípio da minimização da Trusted Computing Base (TCB).

O SC tem controle sobre todos os outros componentes do SCuP, permitindo, dentre outros, o Mecanismo de Inspeção/Introspecção Profunda (MIP) e a Sequência de Boot Seguro.

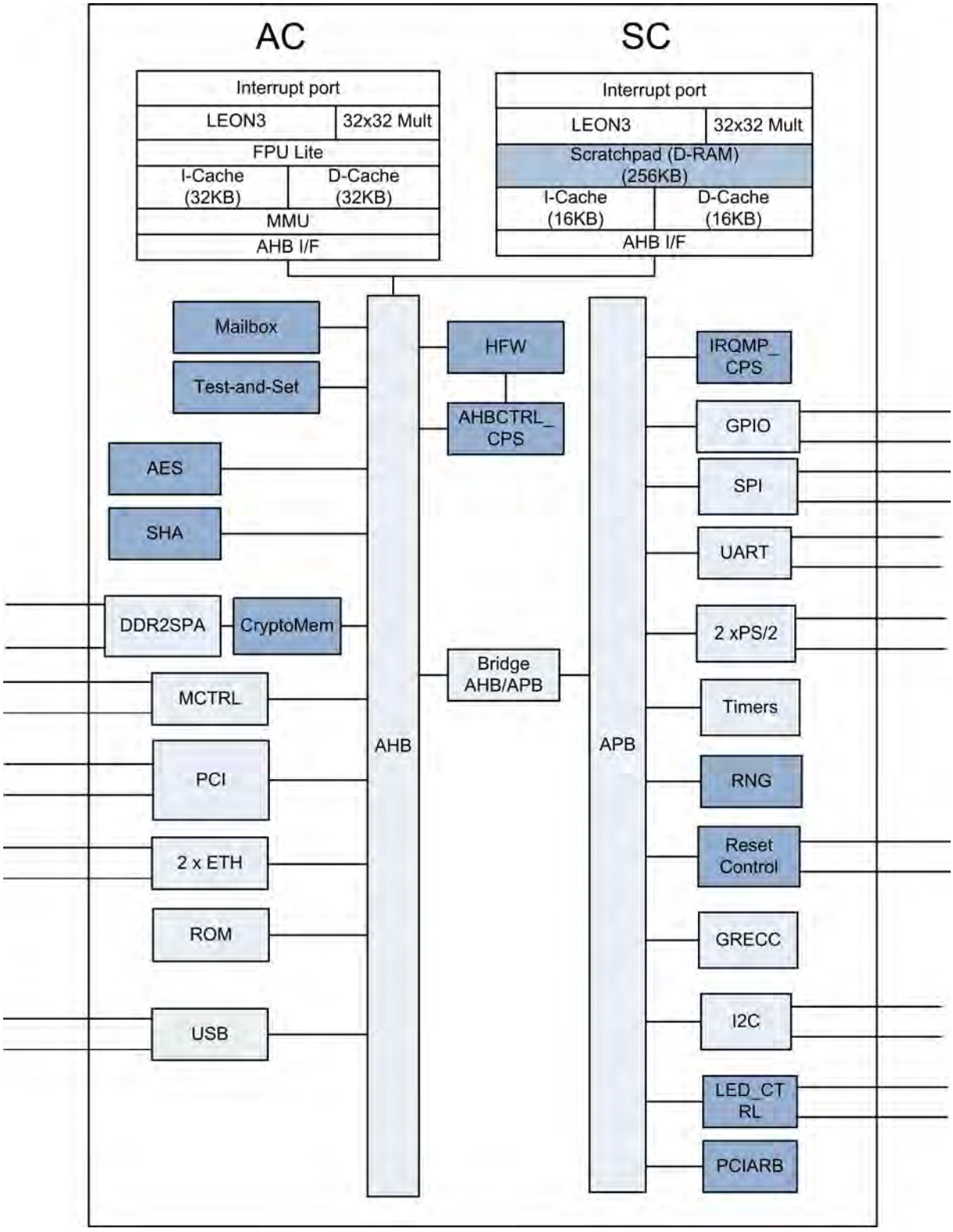

Figura 1. A arquitetura do SCuP mostrando os seus diversos módulos e periféricos.

3.1.2. O Firewall de Hardware

O Firewall de Hardware (HWF) permite que o SC controle o acesso dos mestres de barramentos internos do SCuP aos periféricos. Esta funcionalidade tem como principal objetivo impedir que componentes (de software, de hardware) comprometidos tenham acesso a canais de comunicação com dados em claro.

O HWF funciona por meio da programação de múltiplas regras de firewall que contém as seguintes informações: [mestre, faixa-de-endereços, permissões]. Nesta regra, o acesso do "mestre" à "faixa-de-endereços" está sujeita às respectivas "permissões".

Um exemplo de uso é nos casos de protocolos de votação. Se por um lado toda a parte gráfica e de controle de periféricos do software de votação pode ser executado no AC, esta mesma pilha de software poderá conter defeitos que permitam que a entrada do usuário (o voto digitado) vaze ou seja capturado por uma aplicação mal-intencionada. Em nossa arquitetura, a captura do voto e a sua cifração podem ficar por conta do SC, que, programando o HWF, impede o acesso pelo AC ao periférico PS/2 ao mesmo tempo que permite o uso para si. Obviamente a aplicação precisa ser adaptada para este tipo de uso.

O HWF ainda impede que transações malignas, advindas dos periféricos-mestre de barramento externos ao SCuP tenham a possibilidade de acessar regiões de endereçamento não ativamente permitidas, aumentando o nível de segurança também contra ataques externos.

3.1.3. Cifrador de Memória Externa

O cifrador de memória externa (CryptoMem) permite que regiões da memória RAM externa sejam cifrados de maneira automática com grande aceleração por hardware. Isto permite que o processador AC/SC execute código cifrado da memória externa de maneira transparente. As chaves do CryptoMem são diretamente controladas pelo SC, assim como as faixas de endereçamento que devem ser protegidas. O CryptoMem emprega o algoritmo NIST-FIPS PUB 197 - AES - com chaves de 256 bits em modo de operação não-padrão. O CryptoMem tem performance de processamento de 128 bits/ciclo.

3.1.4. Módulo AES-256 e Módulo SHA-512 de Alto Desempenho

O módulo AES implementa o NIST FIPS PUB 197 com chaves de 256 bits nos modos de operação ECB, CTR, CBC e GCM e desempenho de pico de processamento de 8 bits/ciclo. O módulo SHA implementa o NIST FIPS PUB 180-3, com hashes de 512 bits e desempenho de pico $\geq$ 12 bits/ciclo.

Estes blocos operam como aceleradores de hardware internos ao processador e servem aplicações executando tanto no SC como no AC de forma direta ou através da biblioteca criptográfica da plataforma SCuP. Esta biblioteca também faz uso do acelerador de curvas elípticas presente no processador.

3.1.5. Outros Componentes

O SCuP possui diversos outros componentes com funções de segurança, mas que por questões de espaço somente mencionaremos. Um destes componentes é o TRNG do processador, item essencial na operação da maioria dos esquemas criptográficos. O TRNG do SCuP é não determinístico e explora características físicas das portas lógicas convencionais do semicondutor para a geração e com alta entropia. O TRNG será tema de artigo futuro e por enquanto representa segredo industrial.

Outro componente que merece menção é o Mailbox, utilizado para a comunicação entre núcleos. Este componente foi especificamente projetado para permitir que informações entre AC e SC possam ser trocadas de maneira rápida e protegida do acesso externo.

O último componente que mencionamos é o IRQMP-CPS, componente central no MIP. O IRQMP-CPS possui modificações em relação a um gerenciador convencional de requisições (IRQs) de forma que, quando o AC é provocado pelo SC, o primeiro **obrigatoriamente** executa um trecho de código confiado determinado por SC.

3.2. Funcionalidades do SCuP

3.2.1. SBS - Sequência de Boot Seguro

A sequência de boot seguro (SBS) é fundamental para garantir que o estado da plataforma seja conhecido (íntegro) em ambos os núcleos quando o sistema termina a inicialização. Para tanto, utilizamos uma sequência de boot com verificação de assinaturas digitais que vai da BIOS às aplicações de usuário. A sequência é a seguinte:

Etapa/Fase	Nome	Descrição
1	Load/Decode	O software que carrega e decifra o software da ROM externa, estará gravado na ROM interna, e ele utilizará o scratchpad do SC como sua memória RAM
1.1	Auto-testes SC	Realiza auto-testes de funções criptográficas e de integridade interna
1.2	Zeração	Zera todos os buffers e caches internos e a RAM
1.3	Cópia da ROM	Copia o conteúdo da ROM externa para o scratchpad
1.4	Decifra	Decifra o conteúdo carregado no scratchpad
1.5	Verifica	Verificar a assinatura digital do conteúdo do scratchpad
1.6	Carga	Limpa registradores e ajusta o PC para o início do scratchpad
2	Execute	O software da ROM externa (BIOS) está carregado no scratchpad, e utiliza essa memória como RAM
2.1	HFW	Configura o HWF (libera acessos a determinados recursos do sistema)
2.2	SC	Configura o SC
2.3	Imagem AC	Obtém a imagem de boot do Linux (o qual executará no AC). Opcionalmente (a) Verifica hardware, (b) Continua boot pela rede, (c) Acessa a memória externa, (d) Atualiza a ROM externa

2.4	Decifra	Decifra a imagem de boot do Linux
2.5	Verificar	Verifica a imagem de boot do Linux
2.6	Carrega	Carrega a imagem de boot do Linux no endereço inicial do AC
2.7	Libera	Libera o AC (“acorda” o processador AC)
3	Boot Linux	Imagem de boot do Linux carregada, recursos liberados e AC “ acordado”

O mecanismo de boot seguro impede que ataques de substituição/alteração de binários sejam possíveis no SCuP. Tanto quanto pudemos averiguar, o SCuP é o primeiro processador a implementar uma sequência de boot seguro multi-core e também o primeiro a fazer isso com serviços de assinatura digital e sigilo simultaneamente. Entretanto, este mecanismo não é suficiente para proteger contra ataques online, onde defeitos das aplicações são exploradas pelos adversários, como em ataques de estouro de pilha (execução de dados).

Por este motivo, mecanismos de proteção contra ataques em tempo de execução foram incluídos no SCuP, como o MIP.

3.2.2. MIP - Mecanismo de Introspecção/Inspeção Profunda

O objetivo do MIP é permitir que o estado do núcleo AC seja totalmente conhecido e acessível para inspeção completa impedindo que código malicioso se aloje em qualquer parte dos elementos de memória do núcleo. Isto é necessário pois o núcleo AC executa uma pilha extensa de software, com elementos possivelmente não assinados digitalmente (e não verificados pela SBS).

MIP foi inspirado no CoPilot, mas apresenta diversas modificações e vantagens. Em primeiro lugar, o CoPilot não é capaz de ter acesso total ao estado da CPU principal do sistema dado que seu acesso era externo, via PCI, o que também limita o seu desempenho.

O funcionamento do MIP pode ser assim sumarizado:

- Sob requisição do usuário ou de tempos em tempos, o SC inicia o ciclo de verificação;
- para tanto, o SC gera uma interrupção não mascarável de máxima prioridade no AC (via componente IRQMP-CPS);
- o AC imediatamente começa a servir a requisição em um trecho de código fixo em ROM que realiza verificações (V-COM). Por estar em ROM, não pode ser modificado por adversários;
- V-ROM é utilizado então para verificar a assinatura de código adicional escrito por SC (V-RAM) em posição pré-determinada de memória. Se a assinatura for correta, inicia a execução deste novo trecho de código;
- V-RAM é o código que comanda a verificação do sistema seja por inspeção ou por introspecção. No caso da introspecção, no primeiro passo, o AC empilha todos os seus registradores e descarrega a cache (instrução “flush”) e continua executando o “anti-malware”. No caso da inspeção, AC permanece em “stall” e SC realiza toda as verificação;
- finalizado o trecho V-RAM, o AC retorna à execução normal.

Com a arquitetura do SCuP, a verificação realizada no ciclo do MIP é altamente eficiente, uma vez que a comunicação entre o SC e o AC é realizada por meio do barramento interno ao processador. Além disso, a nossa arquitetura também permite que o SC mantenha serviços essenciais ao sistema enquanto o AC passa pelo MIP tarefas como, por exemplo, manutenção de "watchdogs" ou mesmo controles demandados por periféricos de tempo real.

Por meio do uso do HFW e do MIP simultaneamente, nossa arquitetura permite a construção de mecanismos de recuperação dinâmica de kernel e detecção de rootkits de alta eficiência. Para tanto, o SC protege uma região de memória onde mantém uma cópia do kernel saudável de AC. Assim, ao executar o MIP em modo de inspeção, o SC pode facilmente comparar cópias do kernel e restaurar imagens anteriores, uma vez que um adversário em AC é incapaz de corromper tanto o mecanismo de verificação, como as próprias imagens protegidas por SC. Tanto quanto saibamos, o SCuP é o primeiro processador a dar suporte a este tipo de operação integrada.

3.2.3. Outras Funcionalidades

Além dos mecanismos SBS e MIP, o SCuP ainda incorpora o conceito de pacote de execução segura, introduzido pelo IBM Cell e mais tarde formalizado pela proposta do TEM. Um pacote de execução segura é um blob que contém dados e binários assinados e possivelmente cifrados e que são entregues para um núcleo para processamento. Antes de iniciar a execução, este núcleo verifica a assinatura sobre o pacote (e possivelmente o decifra) antes de iniciar a sua execução, em modo exclusivo.

Apesar de baseado nestas arquiteturas, nossa proposta difere de ambas soluções: tanto no Cell como no TEM, as unidades processantes (núcleos) funcionam como escravos de barramento. Isto significa que pacotes de execução segura vindas do ambiente externo não podem ser utilizadas para verificar o estado do sistema como um todo, ou mesmo levá-lo a um estado conhecido.

No SCuP, entretanto, o SC (responsável pela execução dos pacotes seguros) é o mestre do sistema, o que permite que tais pacotes sejam executados em um ambiente controlado (via MIP e HMW), de fato adicionando uma camada de segurança, em vez de ser um mecanismo acessório. O SCuP é o primeiro processador a realizar esta abordagem.

4. Implementação e Resultados

O SCuP foi implementado em VHDL utilizando a licença comercial do Gaisler Leon 3 e simulado e sintetizado com o Quartus II Full para a plataforma alvo de desenvolvimento Altera Stratix III EP3SL200C2 em um kit de desenvolvimento projetado por nós. O sistema completo consumiu 69.438 ALUTs da FPGA e operou a uma frequência máxima de 140MHz.

A Tabela 2 mostra o consumo de elementos dos principais componentes do sistema. Nota-se que os principais consumos são dos componentes criptográficos de alto desempenho, correspondendo por cerca de 52% da área (ALUT) do processador.

Se por um lado o impacto em elementos consumidos é alto, por outro a plataforma se adapta bem quando mais instâncias de AC e SC são inclusas, pois os componentes

Componente	ALUT	%
AC	10,5K	15
SC	6,5K	10
AES 256	7,5K	11
SHA 512	3,5K	5
HWF	2,0K	2
MemCrypt	25,0K	36
TRNG	1K	1
Outros	13,5K	20
Total	69,5K	100

Tabela 2. Consumo de elementos

criptográficos podem ser compartilhados por todos os núcleos.

No quesito desempenho, a implementação das funcionalidades de segurança do SCuP teve pequeno impacto na frequência máxima de operação. Sintetizando um processador sem os componentes de segurança, obtivemos fmax de 150MHz, contra 140MHz de nosso design, uma penalidade de apenas 6,7%.

Os testes de desempenho dos módulos AES-256 e SHA-512 a partir da biblioteca criptográfica da plataforma foram de 380Mbps e 500Mbps, respectivamente, valores bastante altos para a frequência de operação de 140MHz, mostrando pequeno "overhead" de software.

5. Conclusão e Trabalhos Futuros

O SCuP traz uma arquitetura inovadora, desenvolvida levando-se em conta ataques físicos, ataques lógicos (online e off-line) e os inexoráveis defeitos de software (e hardware). Para tanto, além possuir tal arquitetura, no SCuP introduzimos os mecanismos de introspecção/inspeção profunda e firewall de hardware que, em conjunto com os pacotes de execução segura, garantem múltiplas camadas de segurança independentes no processador. O SCuP, portanto, representa uma nova filosofia no projeto de processadores, onde um cenário de ataques ampliado é considerado, resultando em um sistema mais robusto e mais resistente a quebras de segurança de sub-componentes. Os resultados de implementação até o momento apontam para a total viabilidade do processador, com degradação mínima de desempenho (de 150 para 140MHz, -6,7%) e custos moderados em termos de área (53%). A difusão em silício, esperada para os próximos meses, permitirá que números ajustados de desempenho sejam obtidos e que figuras de desempenho globais sejam estabelecidas, inclusive com o SBS, o MIP e o PES.

Os trabalhos futuros estarão centrados no desenvolvimento das bibliotecas de software e aplicações para uso do SCuP em diversos cenários, desde votação eletrônica, até aviônica.

Referências

[1] Gianpiero Cabodi, Sergio Nocco, and Stefano Quer. Improving SAT-based Bounded Model Checking by Means of BDD-based Approximate Traversals. In *in Design, Automation and Test in Europe, 2003*, pages 898–903, 2003.

[2] Benjie Chen and Robert Morris. Certifying program execution with secure processors. In *HOTOS'03: Proceedings of the 9th conference on Hot Topics in Operating Systems*, pages 23–23, Berkeley, CA, USA, 2003. USENIX Association.

[3] Victor Costan, Luis F. Sarmenta, Marten van Dijk, and Srinivas Devadas. The Trusted Execution Module: Commodity General-Purpose Trusted Computing. In *CARDIS '08: Proceedings of the 8th IFIP WG 8.8/11.2 International Conference on Smart Card Research and Advanced Applications*, pages 133–148, Berlin, Heidelberg, 2008. Springer-Verlag.

[4] Joan G. Dyer, Mark Lindemann, Ronald Perez, Reiner Sailer, Leendert van Doorn, Sean W. Smith, and Steve Weingart. Building the IBM 4758 secure coprocessor. *Computer*, 34(10):57–66, 2001.

[5] Roberto Gallo, Henrique Kawakami, and Ricardo Dahab. On device identity establishment and verification. In *Proc of EuroPKI'09 Sixth European Workshop on Public Key Services, Applications and Infrastructures*, September 2009.

[6] Roberto Gallo, Henrique Kawakami, Ricardo Dahab, Rafael Azevedo, Saulo Lima, and Guido Araujo. A hardware trusted computing base for direct recording electronic vote machines. Austin, Texas, USA, 2010. ACM.

[7] Warren A. Hunt. Mechanical mathematical methods for microprocessor verification. In Rajeev Alur and Doron A. Peled, editors, *Computer Aided Verification*, volume 3114 of *Lecture Notes in Computer Science*, pages 274–276. Springer Berlin - Heidelberg, 2004.

[8] Hiroaki Iwashita, Satoshi Kowatari, Tsuneo Nakata, and Fumiyasu Hirose. Automatic test program generation for pipelined processors. In *Proceedings of the 1994 IEEE-ACM international conference on Computer-aided design*, ICCAD 94, pages 580–583, Los Alamitos, CA, USA, 1994. IEEE Computer Society Press.

[9] Ruby B. Lee, Peter C. S. Kwan, John P. McGregor, Jeffrey Dwoskin, and Zhenghong Wang. Architecture for protecting critical secrets in microprocessors. *SIGARCH Comput. Archit. News*, 33:2–13, May 2005.

[10] A Mirjalili, S, and Lenstra. Security Observance throughout the Life-Cycle of Embedded Systems. In *International Conference on Embedded Systems*, 2008.

[11] NIST. *Security requirements for cryptographic modules, Federal Information Processing Standards Publication (FIPS PUB) 140-2*, 2002.

[12] E. Oksuzoglu and D.S. Wallach. VoteBox Nano: A Smaller, Stronger FPGA-based Voting Machine (Short Paper). *usenix.org*, 2009.

[13] John P., Dwoskin, and Ruby B. Lee. A framework for testing hardware-software security architectures. Austin, Texas, USA, 2010. ACM.

[14] Bryan D. Payne, Martim Carbone, Monirul Sharif, and Wenke Lee. Lares: An architecture for secure active monitoring using virtualization. In *Proceedings of the 2008 IEEE Symposium on Security and Privacy*, pages 233–247, Washington, DC, USA, 2008. IEEE Computer Society.

[15] Nick L. Petroni, Jr., Timothy Fraser, Jesus Molina, and William A. Arbaugh. Copilot - a coprocessor-based kernel runtime integrity monitor. In *Proceedings of the 13th*

conference on USENIX Security Symposium - Volume 13, SSYM'04, pages 13–13, Berkeley, CA, USA, 2004. USENIX Association.

[16] Sandip Ray and Warren A. Hunt. Deductive verification of pipelined machines using first-order quantification. In Rajeev Alur and Doron A. Peled, editors, *Computer Aided Verification*, volume 3114 of *Lecture Notes in Computer Science*, pages 254–256. Springer Berlin - Heidelberg, 2004.

[17] Naveen K Sastry. *Verifying security properties in electronic voting machines*. PhD thesis, University Of California, Berkeley, 2007.

[18] F. Schneider, Greg Morrisett, and Robert Harper. A language-based approach to security. In *Informatics*, pages 86–101. Springer, 2001.

[19] K. Shimizu, H. P. Hofstee, and J. S. Liberty. Cell broadband engine processor vault security architecture. *IBM J. Res. Dev.*, 51(5):521–528, 2007.

[20] G. Edward Suh, Charles W. O'Donnell, and Srinivas Devadas. Aegis: A single-chip secure processor. *IEEE Design and Test of Computers*, 24(6):570–580, 2007.

[21] The Common Criteria Recognition Agreement. Common criteria for information technology security evaluation v3.1 revision 3, July 2009.

[22] Trusted Computing Group. *Trusted Platform Module Main Description Level 2 version 1.2 revision 116*, March 2011.

[23] Paul D. Williams. *CuPIDS: increasing information system security through the use of dedicated co-processing*. PhD thesis, West Lafayette, IN, USA, 2005. AAI3191586.