Instituto de Computação
Universidade Estadual de Campinas

Um *Framework* para Desenvolvimento e Implementação de Sistemas Seguros Baseados em Hardware

Roberto Alves Gallo Filho

Este exemplar corresponde à redação final da Tese devidamente corrigida e defendida por Roberto Alves Gallo Filho e aprovada pela Banca Examinadora.

Campinas, 15 de Setembro de 2012.

Prof. Dr. Ricardo Dahab (Orientador)

Tese apresentada ao Instituto de Computação, UNICAMP, como requisito parcial para a obtenção do título de Doutor em Ciência da Computação.

FICHA CATALOGRÁFICA ELABORADA POR
ANA REGINA MACHADO - CRB8/5467
BIBLIOTECA DO INSTITUTO DE MATEMÁTICA, ESTATÍSTICA E
COMPUTAÇÃO CIENTÍFICA - UNICAMP

Gallo Filho, Roberto Alves, 1978-
G137f Um framework para desenvolvimento e implementação de sistemas seguros baseados em hardware / Roberto Alves Gallo Filho. – Campinas, SP : [s.n.], 2012.

Orientador: Ricardo Dahab.
Tese (doutorado) – Universidade Estadual de Campinas, Instituto de Computação.

1. Arquitetura de computador - Medidas de segurança. 2. Proteção de dados. 3. Tecnologia da informação - Medidas de segurança. 4. Redes de computadores - Medidas de segurança. 5. Sistemas de segurança. I. Dahab, Ricardo,1957-. II. Universidade Estadual de Campinas. Instituto de Computação. III. Título.

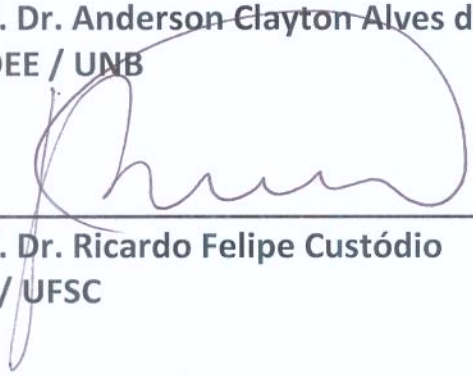
Informações para Biblioteca Digital

Título em inglês: A framework for development and implementation of secure hardware-based systems
Palavras-chave em inglês:
Computer architecture - Security measures
Data protection
Information technology - Security measures
Computer networks - Security measures
Security systems
Área de concentração: Ciência da Computação
Titulação: Doutor em Ciência da Computação
Banca examinadora:
Ricardo Dahab [Orientador]
Anderson Clayton Alves Nascimento
Ricardo Felipe Custodio
Marco Aurélio Amaral Henriques
Sandro Rigo
Data de defesa: 24-08-2012
Programa de Pós-Graduação: Ciência da Computação

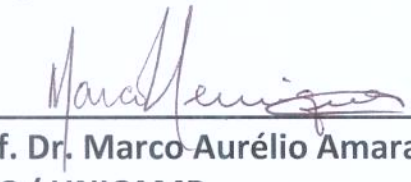
TERMO DE APROVAÇÃO

Tese Defendida e Aprovada em 24 de Agosto de 2012, pela Banca examinadora composta pelos Professores Doutores:

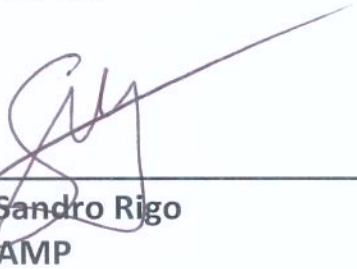
Prof. Dr. Anderson Clayton Alves do Nascimento
FT-DEE / UNB



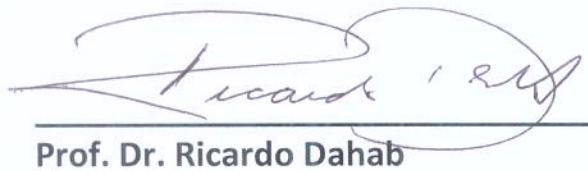
Prof. Dr. Ricardo Felipe Custódio
DIE / UFSC



Prof. Dr. Marco Aurélio Amaral Henriques
FEEC / UNICAMP



Prof. Dr. Sandro Rigo
IC / UNICAMP



Prof. Dr. Ricardo Dahab
IC / UNICAMP

Instituto de Computação
Universidade Estadual de Campinas

Um *Framework* para Desenvolvimento e Implementação de Sistemas Seguros Baseados em Hardware

Roberto Alves Gallo Filho[1]

Agosto de 2012

Banca Examinadora:

- Prof. Dr. Ricardo Dahab (Orientador)
- Prof. Dr. Anderson Clayton Alves Nascimento
  Departamento de Engenharia Elétrica (UnB)
- Prof. Dr. Ricardo Felipe Custodio
  Departamento de Informática e Estatística (UFSC)
- Prof. Dr. Marco Aurélio Amaral Henriques
  Faculdade de Engenharia Elétrica e de Computação (UNICAMP)
- Prof. Dr. Sandro Rigo
  Instituto de Computação (UNICAMP)

[1]Suporte financeiro de: Bolsa CNPq 2005 a 2007, Bolsa SERASA-Experian, Bolsas FUNCAMP dos projetos Banco do Brasil e Receita Federal do Brasil, despesas projeto temático FAPESP e subvenção econômica FINEP

Resumo

A concepção de sistemas seguros demanda tratamento holístico, global. A razão é que a mera composição de componentes individualmente seguros não garante a segurança do conjunto resultante[2].

Enquanto isso, a complexidade dos sistemas de informação cresce vigorosamente, dentre outros, no que se diz respeito: i) ao número de componentes constituintes; ii) ao número de interações com outros sistemas; e iii) à diversidade de natureza dos componentes. Este crescimento constante da complexidade demanda um domínio de conhecimento ao mesmo tempo multidisciplinar e profundo, cada vez mais difícil de ser coordenado em uma única visão global, seja por um indivíduo, seja por uma equipe de desenvolvimento.

Nesta tese propomos um *framework* para a concepção, desenvolvimento e *deployment* de sistemas baseados em hardware que é fundamentado em uma visão única e global de segurança. Tal visão cobre um espectro abrangente de requisitos, desde a integridade física dos dispositivos até a verificação, pelo usuário final, de que seu sistema está logicamente íntegro.

Para alcançar este objetivo, apresentamos nesta tese o seguinte conjunto de componentes para o nosso *framework*: i) um conjunto de considerações para a construção de modelos de ataques que capturem a natureza particular dos adversários de sistemas seguros reais, principalmente daqueles baseados em hardware; ii) um arcabouço teórico com conceitos e definições importantes e úteis na construção de sistemas seguros baseados em hardware; iii) um conjunto de padrões (*patterns*) de componentes e arquiteturas de sistemas seguros baseados em hardware; iv) um modelo teórico, lógico-probabilístico, para avaliação do nível de segurança das arquiteturas e implementações; e v) a aplicação dos elementos do *framework* na implementação de sistemas de produção, com estudos de casos muito significativos[3]. Os resultados relacionados a estes componentes estão apresentados nesta tese na forma de coletânea de artigos.

[2]Técnicas "greedy" não fornecem necessariamente os resultados ótimos. Mais, a presença de componentes seguros não é nem fundamental.

[3]Em termos de impacto social, econômico ou estratégico

Abstract

The conception of secure systems requires a global, holistic, approach. The reason is that the mere composition of individually secure components does not necessarily imply in the security of the resulting system[4].

Meanwhile, the complexity of information systems has grown vigorously in several dimensions as: i) the number of components, ii) the number of interactions with other components, iii) the diversity in the nature of the components. This continuous growth of complexity requires from designers a deep and broad multidisciplinary knowledge, which is becoming increasingly difficult to be coordinated and attained either by individuals or even teams.

In this thesis we propose a framework for the conception, development, and deployment of secure hardware-based systems that is rooted on a unified and global security vision. Such a vision encompasses a broad spectrum of requirements, from device physical integrity to the device logical integrity verification by humans.

In order to attain this objective we present in this thesis the following set of components of our framework: i) a set of considerations for the development of threat models that captures the particular nature of adversaries of real secure systems based on hardware; ii) a set of theoretical concepts and definitions useful in the design of secure hardware-based systems; iii) a set of design patterns of components and architectures for secure systems; iv) a logical-probabilistic theoretical model for security evaluation of system architectures and implementations; and v) the application of the elements of our framework in production systems with highly relevant study cases. Our results related to these components are presented in this thesis as a series of papers which have been published or submitted for publication.

[4]Greedy techniques do not inevitably yield optimal results. More than that, the usage of secure components is not even required

Dedicatória

Em saudosa memória de Arnaldo Gallo.

Agradecimentos

Algumas jornadas possuem caminhos diretos e rápidos. Certamente, este não foi o meu caso; gerir uma empresa, casar-se e mudar de direção de pesquisa são eventos que para muitos implicaria em desistir de caminhar em direção ao doutoramento. Este, felizmente, também não foi o meu caso; tenho muito e a muitos para agradecer.

Inicio agradecendo aos contribuintes, ao Estado de São Paulo e à própria UNICAMP que investiram pesadamente em minha formação, desde a graduação até hoje. Formar um engenheiro, um mestre e um doutor é dispendioso e demanda um grande esforço da sociedade em um país com muitas carências[5]. Obrigado.

Agradeço aos meus sócios Henrique Kawakami e Leonardo Cabral que durante muitos anos permitiram que eu me dedicasse à pesquisa mesmo nos momentos críticos, tendo eles que se desdobrarem. Agradeço em especial ao Henrique por sempre estar disposto a discutir e ajudar a amadurecer as ideias, por mais insólitas que parecessem no início. Obrigado.

Agradeço ao meu orientador Ricardo Dahab que foi o grande divisor de águas no meu doutorado. Apesar de eu ter ingressado na graduação em 1997, só descobri o que é um orientador na acepção completa da palavra em 2009, ano que posso dizer que de fato comecei o meu doutoramento. Obrigado.

Agradeço à minha amada esposa Fernanda Gallo pelo apoio irrestrito; com viagens constantes a trabalho ou acadêmicas, tempo escasso durante a semana em função do trabalho e dos estudos e finais de semana pesquisando, Fernanda sempre esteve me motivando. Muito obrigado.

Agradeço à minha família que antes de tudo entende a importância da formação na vida da pessoa. Mais do que isso, família onde um ambiente de análise e de crítica construtiva sempre esteve presente. Obrigado mãe, pai, vovó, vovô e queridas irmãs.

[5] Certamente este esforço se paga

Sumário

Lista de Tabelas

Lista de Figuras

Capítulo 1

Introdução

1.1 Motivação

Descrição do Problema

Problemas de segurança afligem sistemas computacionais há décadas. Nos últimos dez anos as ameaças, antes consideradas simples, evoluíram de casos ocasionais e gerenciáveis para um cenário onde os ataques são complexos, engenhosos e de alto risco. Essa evolução seguiu um processo natural e interativo entre "atacantes" e "defensores": os primeiros sempre procurando pelo elo mais fraco (a falha de segurança mais barata) para ser explorado, e os defensores tentando tornar os elos atingidos ao menos tão resistentes quanto os demais que formam as cadeias de segurança.

Como segurança em si não é o objetivo mais comum de um sistema computacional, mas sim uma propriedade desejada (espera-se que um sistema conforme-se a uma dada política de segurança, caso exista uma), a evolução das ameaças e contramedidas em sistemas computacionais estão sujeitas à corrida frenética do trio "desempenho versus preço versus *features*" [40], que deixa pouco espaço para melhorias mais drásticas de segurança. Neste cenário, contramedidas típicas de segurança são pensadas e implementadas de forma localizada e minimalista, estabelecendo uma colcha de retalhos sem um *framework* geral e coesivo [3, 48].

Mesmo quando os sistemas são pensados do zero [15, 63, 60, 32], a visão holística necessária para a segurança é um obstáculo a ser vencido. Esta dificuldade pode ser atribuída, em grande parte, ao amplo espectro de áreas do conhecimento envolvidas na segurança de sistemas computacionais. Estas áreas incluem Matemática Discreta, Estatística, Eletrônica, Arquitetura de Computadores, Interfaces Homem-máquina e Física.

As Particularidades da nossa Visão do Problema

Computação confiável, em ambientes reais, práticos, demanda mais do que simplesmente primitivas criptográficas, arquiteturas para execução segura de código, políticas de segurança ou provas de modelos. Demanda também a inclusão e reconhecimento de outros fatores mais práticos.

Em primeiro lugar, a admissão de que sistemas seguros são frequentemente quebrados, seja por avanços técnicos, seja por falhas de projeto, falhas de modelagem ou falhas de implementação. *Para todos os efeitos a quebra de segurança de um sistema não trivial é certa. Resta saber quando, com qual impacto.* Reconhecemos esta característica como tendo uma natureza probabilística: "qual a chance deste componente ser subvertido em/-com quanto(s) tempo/recursos?". Esta visão é bastante diferente daquela correntemente empregada nos trabalhos desta área.

Em segundo lugar, deve-se levar em conta que, em algum ponto (direta ou indiretamente), um sistema seguro tem como interface ou interessado um ou mais humanos (ou entidades), que confiam naquele sistema com algum grau de certeza (e medir essa confiança não é simples). Utilizar desavisadamente um sistema verificador para checar um sistema em verificação, uma saída que parece razoável em um primeiro momento, na verdade, de maneira contra-intuitiva, tipicamente *diminui* o nível de confiança global ao invés de aumentá-lo. Por isso, pensar a interação "usuários-sistemas" com atenção é fundamental.

Um terceiro fator de grande importância é econômico, já que sistemas reais têm custos financeiros reais: "trade-offs" e balanceamento do nível de segurança envolvem naturalmente todos os componentes do sistema, mas não somente eles; envolvem também os custos humanos daqueles que fazem interface com o sistema. Manter o conceito de balanceamento dos custos, tipicamente significa deixar de procurar a solução de segurança hermética . Sistemas eleitorais são um exemplo onde o balanceamento de custos tem enorme importância.

O quarto item que torna a nossa visão do problema distinta daquela usualmente encontrada na literatura é a percepção de que o emprego de qualquer sistema seguro está imerso em um *ciclo de vida* amplo, não sendo composto somente por *setup* e utilização. Fazem parte do ciclo de vida de um sistema seguro baseado em hardware: i) concepção; ii) manufatura; iii) logística; iv) *setup*; v) uso; vi) manutenção; e vii) desativação (por obsolescência, fim de utilidade ou comprometimento).

1.2 Objetivos desta Tese

Aqui não temos a presunção de propor uma teoria unificada de segurança computacional mas, sim, contribuir com os elementos-chave de um *framework* baseado em hardware para arquiteturas de sistemas seguros sob uma *visão unificada de segurança*.

No caminho da construção deste *framework*, nós não só colhemos e utilizamos conjuntos e regras de segurança bem estabelecidos, como também fizemos avanços. Propusemos também novos conceitos, sempre que possível implementando os novos elementos em sistemas de produção de alta relevância.

Portanto, o nosso objetivo mais amplo com o *framework* é permitir a concepção, desenvolvimento e o emprego de sistemas baseados em hardware que possuam objetivos de segurança considerados equilibradamente no seu conjunto e concebidos durante o projeto do sistema; isto é, sistemas seguros por construção.

1.3 Estado da Arte em Segurança de Sistemas em Hardware

Este *framework* é relacionado com, entre outras, as áreas de: i) co-processadores criptográficos; ii) execução segura de código; iii) arquiteturas para execução monitorada; iv) integridade de sistemas; e v) verificação segura de sistemas e padrões.

1.3.1 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 [15], 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, como em aplicações embarcadas. Neste sentido, a arquitetura AEGIS [63] 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 modificações de arquitetura em um núcleo 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 [11] 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*.

1.3.2 Execução Segura de Código

As aplicações de DRM estão sujeitas a um modelo de ameaças (*threat model*) particularmente interessante: embora o conteúdo (aplicações, músicas, filmes) possa 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.

Este modelo de ameaças foi aquele utilizado na concepção da geração atual do padrão do TPM do Trusted Computing Group [66], 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 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 [13] propuseram e implementaram (utilizando um processador Javacard) o TEM, capaz de executar código seguro no próprio módulo, através de 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 [60], 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 SPEs, 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*[1].

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 [32, 48]. 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 existente, 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 que provê serviços de confidencialidade e atestação remotos.

A especificação do ARM TrustZone (TZ) [7] descreve uma e arquitetura relacionada para execução segura de código como serviço provido pelo sistema. Antes proprietária, a API foi liberada pela ARM em 2009 e utiliza o conceito de serviço seguros, providos por meio de um "mundo seguro" a um "mundo normal". A API, no entanto, é bastante flexível e os reais componentes e mecanismos de segurança empregados dependem da implementação específica.

O conjunto de tecnologias da Intel chamado IPT a ser lançado durante o ano de 2012, deverá trazer um conjunto de mecanismos de segurança para apoiar a execução segura de software, dentre eles i) um *token* criptográfico embarcado no próprio processador, ii) extensões à instrução de execução segura e iii) canais de E/S seguros, em especial, com a apresentação de uma interface de vídeo cifrada, garantindo comunicação segura entre sistemas remotos e usuários finais dos sistemas computacionais.

[1]Modo de execução seguro e isolado do processador

1.3.3 Arquiteturas para Execução Monitorada

Apesar de não apontado pelo trabalho de Lee [32], 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 que o sistema que faz a verificação (ou sistema verificador) não sofre das mesmas fragilidades que o sistema em verificação, 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 [58, 32] 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, até onde sabemos, nenhum trabalho tem se atentado 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 [72] e CoPilot [50], 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* [58]). 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.

1.3.4 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 [56]. Quaisquer tentativas de se atingir estes objetivos têm implicações diretas na concepção das máquinas de votação (*DRE Voting Machines*).

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 [47], assim como em nossos resultados [17, 20], utilizando a noção de *caminhos confiados* e *classe de nível de confiança*, a última introduzida por nós.

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 aponta Mirjalili e Lenstra [40]. Neste sentido, padrões como o FIPS 140-2 [46] e o Common Criteria [65] 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).

O conceito de fusão de segurança (*security fusion*) [43] também é um exemplo relevante de esforço no sentido de alcançar-se segurança de forma construtiva: a composição de componentes individualmente inseguros em sistemas seguros.

1.3.5 Verificação dos Sistemas e Padrões

O Common Criteria (CC), por sua vez, é um meta-padrão, que define *templates* sobre os quais perfis de segurança (*security profiles*) devem ser elaborados, e que descrevem 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 [23, 27, 51], dificultando o seu uso na prática. Desta forma, técnicas totalmente informais ou mistas são utilizadas na verificação das propriedades do sistemas [9]. Neste sentido, técnicas de verificação lógica de segurança baseadas em simulação, em especial via introspeção de máquinas virtuais, têm se mostrado úteis, como mostram os trabalhos de Payne [49] e Dwoskin [48].

1.4 Metodologia Utilizada

Para alcançar os objetivos esperados do nosso *framework*, a nossa metodologia se baseou nos seguintes itens:

1. Definição de um modelo de ataques que capture a natureza peculiar dos adversários de sistemas seguros reais, principalmente daqueles baseados em hardware.;
2. Estabelecimento de um arcabouço teórico, com fundamentos e conceitos particularmente úteis na construção de sistemas seguros baseados em hardware;
3. Criação de uma coleção de padrões (*patterns*) de componentes e arquiteturas de sistemas seguros baseados em hardware;
4. Um modelo lógico probabilístico para avaliação do nível de segurança das arquiteturas e implementações;
5. Aplicação dos elementos do *framework* na implementação de sistemas de produção em casos muito significativos (de impacto social, econômico ou estratégico).

Estes itens refletiram o nosso modo de trabalho. Entretanto, em vez de tentar construir cada um destes elementos cronologicamente para posteriormente avançar para o próximo (de maneira puramente sequencial), preferimos uma estratégia iterada: i) selecionamos um problema de interesse; ii) o revisitamos com um cenário de ataques ampliado; iii) ampliamos ou propomos a base teórica necessária ao seu estudo; iv) ampliamos ou propomos novas construções para atacar o problema; v) observamos o nível de segurança com um enfoque probabilístico; e v) implementamos essas construções em casos reais.

Esta metodologia tem diversas vantagens: permitiu o refinamento das nossas construções de segurança e ilustrou o caminho natural pelo qual um leitor interessado em usar o *framework* percorreria ao defrontar-se com as vulnerabilidades do seu próprio sistema.

1.4.1 Modelo de Ataques

Existe um equilíbrio natural entre a abrangência dos modelos de ataque e a viabilidade de sua análise. Modelos de ataques demasiadamente amplos tendem a impossibilitar a obtenção de provas de segurança, em função da complexidade adicionada. Por outro lado, modelos de ataques reducionistas podem deixar características importantes dos sistemas fora da análise, e como resultado levar a ataques sutis mas não menos sérios, como os ataques por canais colateriais [1, 28, 54] e ataques por vias subliminares [16].

Na revisão bibliográfica que fizemos estudamos diversos modelos de ataques utilizados em outros trabalhos de hardware seguro para fins diferentes (DRM, proteção de chaves criptográficas (HSMs, *tokens*), votação eletrônica, e execução segura de código), e analisamos possíveis falhas de segurança existentes nos sistemas quando um modelo essencialmente prático, amplo, é utilizado na análise. Em nossos trabalhos já publicados ou submetidos, pudemos também apresentar modelos de ataques ampliados em consonância com a visão de nosso *framework*. Muitos destes cenários podem ser capturados pelo nosso modelo lógico-probabilístico, o FORTUNA.

1.4.2 Arcabouço Teórico

A caracterização de métodos, ações, componentes e definições é fundamental para a descrição e criação de um modelo mais geral de plataforma segura e para a facilitar a aplicação dos métodos lógico-probabilísticos que empregamos neste *framework*.

Durante a revisão bibliográfica realizada estivemos atentos aos fundamentos utilizados pelos demais trabalhos, colhendo itens de aplicabilidade ampla e geral que poderiam ser ou foram utilizados em mais de uma ocasião. Um dos itens comumente encontrados é a *chave mestra de dispositivo* (ou *device master key - DMK*), utilizada em diversos trabalhos como raiz de confiança sobre a qual a proteção de algum outro parâmetro (dados, código, outro material de chaves) está baseada. Como resultado desta revisão bibliográfica pudemos conceber, aplicar e contribuir com alguns conceitos-chave (com resultados obtidos em material já publicado), validando a nossa linha de trabalho em hardware seguro. Fomos capazes também de capturar muitas destas construções com o nosso modelo lógico-probabilístico.

1.4.3 *Patterns*

Enquanto os fundamentos contêm elementos que podem ser empregados, ou devem ser observados, virtualmente por qualquer sistema seguro, existem outras construções cujo uso é delimitado ou guiado pela aplicação-alvo do sistema. A catalogação e emprego de *patterns* é comum na área de Engenharia de Software (em especial design *patterns*), mas muito escasso na área de hardware seguro, em especial quando *architectural patterns*[2] são considerados, talvez porque a área ainda esteja em evolução acelerada.

Como *patterns* são mais específicos, não procuramos ser exaustivos na catalogação; ou, de outra forma, durante a revisão bibliográfica, para cada uma das áreas-problema (ou casos) abarcadas, procuramos o estado da arte das soluções ali propostas para só então propor novas e melhores soluções dado o nosso cenário de ataques. Estes *patterns* estão presentes nos demais capítulos deste documento e um sumário deles está disposto mais adiante neste capítulo.

1.4.4 Modelo Lógico-Probabilístico

Sistemas reais são compostos por múltiplos módulos com correlações complexas, algumas vezes ocultas. Sistemas reais são frequentemente quebrados, muitas vezes por técnicas criadas especialmente para esse objetivo. A qualquer momento, a integridade de um sistema pode ser em última instância avaliada por humanos. Estas características imprimem uma natureza com conotação ao mesmo tempo probabilística e lógica ao problema[3]. Se adicionamos o fato de que sistemas possuem custos reais, fica claro que o balanceamento das medidas de segurança na concepção de sistemas é fundamental e apresenta complexidade considerável.

Uma vez que provas de segurança formais para sistemas compostos de hardware e software são NP-difíceis ou indecidíveis para o caso geral, heurísticas são frequentemente utilizadas como abordagem de testes e desenvolvimento de projetos. Heurísticas, no entanto, possuem o inconveniente de requerer indivíduos com amplo conhecimento técnico e experiência nas áreas correlatas.

Em nossos resultados procuramos capturar de forma mais precisa (e menos heurística) estas características, o que levou à apresentação dos Esquemas de Amplificação de Confiança (EuroPKI2009) [17] e à Classificação de Vulnerabilidades dos Canais de Verificação (ACSAC2010) [20], culminando no framework lógico probabilístico FORTUNA (NSS2011) [18].

[2]Architectural patters são descrições de mais alto nível (mais amplas) do que *design patterns*. Respondem mais à pergunta "o que?" do que "como?'.

[3]A característica probabilística advém da própria natureza de um sistema físico se suas interações, sujeito a imprecisões em todos os níveis, dos transistores ao comportamento dos adversários.

1.4.5 Aplicação dos Elementos do *Framework*

A viabilidade e validade de nossas propostas sempre que possível passou por uma confirmação prática, com sistemas significativos e adversários reais. Entendemos que a aplicação prática de alta escala, apesar de nem sempre possível, é importante pois funciona como validador de nossas propostas nos quesitos i) custos versus nível de segurança, ii) validação de *patterns*.

Desta forma, para cada novo sistema incluído em nosso *framework* (técnica iterada ao invés de puramente sequencial), o ciclo metodológico foi re-exercitado, levando à depuração do ciclo e à expansão dos componentes do *framework* (modelo de ataques, bases, *patterns*, modelo probabilístico, casos práticos) resultando nesta tese.

1.5 Contribuições desta Tese

1.5.1 Síntese dos Resultados Obtidos

Nos artigos já publicados, alcançamos tanto novos resultados teóricos e conceituais como resultados práticos que compõem o nosso *framework*. No artigo apresentado no 6o. EuroPKI em 2009, publicado no LNCS da Springer, com título *"On Device Identity Establishment and Verification"*, capítulo 2 desta tese, introduzimos diversos conceitos, dentre eles o *CID*, sua ligação com o ciclo de vida de sistemas seguros, sua integridade física e lógica, e a interação com usuários por meio de um esquema de amplificação de confiança.

Os conceitos ali apresentados foram implementados em um HSM real para experimentação. Boa parte dos resultados foram incorporados na versão de produção do equipamento ASI-HSM [37, 17, 30], que hoje contém, gerencia e protege as chaves da AC Raiz da Infraestrutura de Chaves Públicas Brasileira (AC-Raiz da ICP-Brasil), AC-Receita Federal, AC-SERPRO e AC-NIB.BR/DNSSEC, dentre outras ACs, com histórico de segurança perfeito. O ASI-HSM foi o primeiro HSM homologado para uso na ICP-Brasil e é o único no mais alto nível de segurança previsto: NSF2-NSH3.

No artigo aceito e apresentado no 26o ACSAC em dez/2010, intitulado *"T-DRE: A Hardware Trusted Computing Base for Direct Recording Electronic Vote Machines"* (publicado pela ACM), capítulo 3 desta tese, nós avançamos nos resultados do artigo do EuroPKI'09, utilizando e estendendo os resultados conceituais anteriormente introduzidos, além de propor novos componentes para o nosso *framework*: a análise da segurança dos canais de verificação humana, e uma nova arquitetura de segurança com controle de acesso a periféricos controlados por hardware com privilégios baseados em certificados digitais.

As novidades apresentadas foram implementadas em um protótipo de bancada de urna eletrônica e mais tarde incorporadas em sua maioria à Nova Urna Eletrônica Brasileira (modelo 2010-2011), com 400 mil equipamentos fabricados (em contratos de mais de

R$400 milhões entre TSE e Diebold-PROCOMP). Deste total, 200 mil urnas já utilizadas na eleição brasileira nacional de 2010. Nenhum caso de fraude foi apurando nestas eleições. Estes fatos dão amplo e sólido suporte às nossas propostas[4].

No artigo aceito e apresentado no 5o NSS em set/2011, intitulado *"FORTUNA – A Probabilistic Framework*[5] for Early Design Stages of Hardware-Based Secure Systems" (publicado pela IEEE), capítulo 4 desta tese, nós estabelecemos os fundamentos teóricos do nosso framework baseado em um conjunto de observações e propriedades. Este conjunto foi utilizado no estabelecimento de três modelos lógico-probabilístico: i) baseado em grafos para relações de proteção, ii) baseado em grafos para modelagem entrópica, iii) baseado em linguagem lógico-probabilística de uso geral.

Sob estes modelos, políticas de segurança como "mínimos privilégios" foram provadas enquanto outras, como "minimização da base computacional confiada" foram desafiadas. Com o modelo baseado em linguagem, construímos uma ferramenta de análise de segurança de sistemas utilizado como suporte no desenvolvimento do *SCuP* – um processador seguro. No nosso entendimento, FORTUNA é o primeiro, e até o momento único, framework para concepção e desenvolvimento inicial de hardware seguro.

No artigo aceito e apresentado no 11o SBSeg em nov/2011, intitulado *"SCuP - Secure Cryptographic Microprocessor"* (publicado nos anais do SBSeg 2011), capítulo 5 desta tese, nós apresentamos o SCuP, um 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 MIP combinados com os PES. O SCuP foi validado em simulações e em FPGA e seguirá para para difusão semicondutora nos próximos meses[6]. O SCuP reúne em um único sistema-alvo grande parte dos elementos de nosso framework.

No artigo submetido para o Journal of Systems and Software em julho/2012, intitulado *"Hardware Security: Towards a Holistic View (or Extending FORTUNA)"*, constante do capítulo 6, nos revalidamos e estendemos a contribuição do FORTUNA ao considerar agora

[4]As recentes fragilidades de software e de processo de desenvolvimento encontradas por Aranha [6] na urna eletrônica brasileira de modelo prévio à nossa contribuição corroboram a necessidade dos mecanismos de segurança por nós propostos. Sem uma raiz de confiança adequada em hardware é inviável atingir segurança sistêmica, mesmo com um software sem defeitos. Com o gerador de números aleatórios em hardware (TRNG), a gestão de chaves pelo módulo criptográfico em hardware (MSM), a sequência de boot seguro com raiz de confiança em hardware (para sigilo e integridade) e a gestão de permissões via certificação digital apresentamos as primitivas sobre as quais a maior parte dos problemas reportados são resolvidos na geração mais atual da urna.

[5]*Framework* no contexto deste artigo é somente um sub-conjunto daquele desta tese, que é mais amplo e inclui, como dito, *patterns*, casos de uso, etc.

[6]Este projeto tem o apoio do programa de subvenção econômico da FINEP, do governo brasileiro.

de maneira explicita a dimensão temporal na análise de sistemas seguros. Em especial, introduzimos métricas para avaliação de elementos dos sistemas-alvo mais propensos a causar problemas de segurança por meio da ferramenta ETC.

O uso da ferramenta ETC serviu de guia para encontrarmos uma fragilidade arquitetural inédita nas especificações dos processadores SPARC V8 e V9. Em uma implementação de prova de conceito do *exploit* pudemos recuperar informações (centenas de bits) entre processos sem uso de código privilegiado. Esta descoberta valida o FORTUNA como ferramenta de auxílio na concepção e projeto de sistemas seguros.

Outras Contribuições

Contribuições na área de segurança não diretamente relacionadas com este framework incluem:

O artigo submetido ao ICITCS 2012 em setembro/2012, intitulado "Unifying Auditing Through HSM Life-Cycle" que discute o problema da auditoria de módulos de segurança criptográfico em um ambiente de infraestrutura de chaves públicas durante todo o seu ciclo de vida, desde o projeto até o descarte. Este artigo foi realizado em colaboração com o LabSEC/UFSC, sendo André Bereza o autor principal.

O artigo aguardando submissão, intitulado "Strong Probabilistic Device Identification Based on the Outflow Problem" que introduz um esquema de identificadores unívocos de equipamentos, especialmente voltados para internet banking, resistentes contra clonagem mesmo com a invasão temporária do equipamento de usuário. O esquema em questão tem o uso planejado na solução de Internet Banking do Banco do Brasil, onde uma vez implantado, deverá proteger mais de um bilhão de transações bancarias por ano. É relevante mencionar que o nosso esquema teve patente depositada junto ao INPI.

Outras contribuições relevantes durante o período de doutoramento mas não relacionadas a este framework incluem os seguintes artigos:

IEEE 9th ISSSTA 2006 - Design, Simulation and Hardware Implementation of a Digital Television System: LDPC channel Coding , Tarciano F. Pegoraro, Fábio A. L. Gomes, Renato R. Lopes, Roberto Gallo, José S. G. Panaro, Marcelo C. Paiva, Fabrício C. A. Oliveira, Fábio Lumertz.

Revista Telecomunicações , Codificação LDPC em Sistemas de Televisão Digital, Tarciano F. Pegoraro, Fábio A. L. Gomes, Fábio Lumertz, Renato R. Lopes, Fabrício C. A. Oliveira, Roberto Gallo, Marcelo C. Paiva and José S. Panaro, volume 9, Santa Rita do Sapucaí, MG, Brasil, dezembro de 2006.

1.5.2 Resumo das Contribuições para o Framework

Na tabela seguinte sumarizamos algumas das principais contribuições de nossos trabalhos para a construção do *framework*. A lista não é exaustiva.

Componente do *framework*	**Item - breve descrição**	**Publicação**
Modelo de ataques	Problemática da clonagem, substituição e adulteração: as vulnerabilidades entre as fases de fabricação, logística e uso de dispositivos criptográficos em seu ciclo de vida, ou, "como acreditar na raiz de confiança?"	EuroPKI'09
Modelo de ataques	Problemática da adulteração dos mecanismos de verificação: diferentes mecanismos possuem diferentes resistências contra diferentes níveis de acesso	ACSAC'10
Modelo de ataques	Ataques sobre a arquitetura do TCG-TPM: segurança menor que a esperada	ACSAC'10
Modelo de ataques	Problemática do *stakeholder* com traidores: o problema de se verificar a integridade de um sistema sob ataque de agentes internos	EuroPKI'09
Modelo de ataques	Problemática do vazamento de informação por canais de interação: a possibilidade obtenção de informações por canais colaterais	NSS'11
Modelo de ataques	Problemática da homogeneidade dos componentes facilitando os ataques	NSS'11
Modelo de ataques	Problemática das mudanças temporais de arquitetura e alteração da respectiva superfície de ataque	Journal, 2012
Fundamentos, definições e bases	Secure Device Epoch (SDE) é a selagem dos dispositivos criptográficos em tempo de fabricação	EuroPKI'09
Fundamentos, definições e bases	Cryptographic Identity (CID), a extensão do DMK, e suas características adicionais: criação, unicidade, proteção, verificabilidade e integridade	EuroPKI'09
Fundamentos, definições e bases	Classificação de resistência a adulteração de canais de verificação	ACSAC'10

Fundamentos, definições e bases	As nove observações sobre sistemas seguros: composição por componentes, natureza probabilística, componentes inseguros em sistemas seguros, componentes seguros em sistemas inseguros, natureza física com abstração lógica, impossibilidade da descrição completa, custo de emprego e prêmio do adversário por componente	NSS'11
Fundamentos, definições e bases	As cinco bases derivadas: canal de interação, potencial entrópico, a impedância entrópica, a segurança implícita, a segurança provida. Estas formam as bases de nosso modelo	NSS'11
Fundamentos, definições e bases	Discussão das bases do FORTUNA sob o ponto de vista temporal	JSS
Fundamentos, definições e bases	A característica de entropia "capacitiva" do componentes	JSS
Patterns	Padrões para a construção de S-SVS e TE-SVS de uso prático	ACSAC'10 e EuroPKI'09
Patterns	Ajuste de cerimoniais de autoridades certificadoras	Euro PKI'09
Patterns	A construção típica de um dispositivo CID-enabled	EuroPKI'09
Patterns	O Master Security Module e a validação de cadeia de certificação por dispositivo confiado, mestre de barramento	ACSAC'10
Patterns	Arquitetura de sistema para privilégios (de acesso a periféricos e chaves) na execução segura de código baseada em certificação digital	ACSAC'10
Patterns	Arquitetura de múltiplos núcleos assimétrica segura e respectiva sequência de boot segura	SBSeg'11
Patterns	O *hardware firewall*, dispositivo de controle de acesso de periféricos e regiões de memória	SBSeg'11
Patterns	O mecanismos de inspeção/introspeção profunda (MIP), pelo qual permite detecção de comprometimento e recuperação dinâmica em caso de ataques	SBSeg'11

Modelo Lógico-Probabilísico	O Shared Verification Scheme (SVS), esquema de verificação compartilhado com amplificação de segurança	EuroPKI'09
Modelo Lógico-Probabilístico	Os esquemas Simple-SVS (S-SVS) e Traitor Evidencing-SVS (TE-SVS)	EuroPKI'09
Modelo Lógico-Probabilístico	O Modelo baseado em grafos de vazamento entrópico e o respectivo modelo matemático	NSS'11
Modelo Lógico-Probabilístico	O Modelo baseado em grafos de relações de proteção e o respectivo modelo matemático	NSS'11
Modelo Lógico-Probabilístico	O Modelo baseado em DTProbLog de análise de segurança e a respectiva ferramenta de análise	NSS'11
Modelo Lógico-Probabilístico	Prova positiva da política de "mínimos privilégios" sob o nosso modelo	NSS'11
Modelo Lógico-Probabilístico	Prova negativa da política de "minimização da base de computação" sob o nosso modelo. Sugestão de modificação de política	NSS'11
Modelo Lógico-Probabilístico	Estimadores entrópicos para os componentes, permitindo a identificação de componentes alvo de análise	JSS
Implementação	Parte das tecnologias implementadas noHSM em uso atual pelas Autoridades Certificadoras Raiz da ICP-Brasil, AC-Receita Federal, AC-ICPEDU e AC-TSE	EuroPKI'09
Implementação	Parte das tecnologias implementadas na Nova Urna Eletrônica Brasileira, com mais de 400.000 unidades fabricadas	ACSAC'10
Implementação	Uso da ferramenta FORTUNA no desenvolvimento do processador SCuP	NSS'11
Implementação	Primeiro processador seguro, de uso geral, do hemisfério sul, indo para difusão semicondutora em 2012	SBSeg'11
Implementação	Descoberta e exploração de falha de segurança na arquitetura SPARC V8 e V9 por meio da ferramenta ETC	Journal'12

1.6 Organização deste Documento

Esta tese é organizada na forma de coletânea, em ordem cronológica, de cinco artigos apresentados como capítulos. São apresentados apenas artigos de autoria principal deste autor.

Cada artigo, na forma de capítulo, é autocontido. Como consequência, não há uma ordem obrigatória de leitura, porém recomendamos que seja na ordem cronológica em que se apresenta.

Os artigos estão apresentados em seu conteúdo original de publicação e sofreram apenas modificações de forma para compilação como coletânea. Resulta da coletânea uma pequena redundância na seção 6.2 do capítulo 6 e seção 5.2 do capítulo 5 em relação ao resto do texto desta tese. É seguro não ler as seções em questão caso o texto da tese seja lido na ordem em que se encontram.

Em função dos limites de páginas impostos nos processos de publicação dos artigos, foi feito o uso recorrente de acrônimos nas submissões originais. Para facilitar a leitura, incluímos ao final desta tese um glossário contendo os principais termos.

No último capítulo apresentamos as conclusões e possíveis trabalhos futuros, em adição àqueles contidos nos artigos.

Capítulo 2

On Device Identity Establishment and Verification

Publicação: Este artigo foi apresentado no *"Sixth European Workshop on Public Key Services, Applications and Infrastructures - EuroPKI2009"*, na data de 10 a 11 de Setembro de 2009, em Pisa, Itália e publicado pela Springer na série LNCS, Volume 6391, em 2010.

Roberto Gallo[1,2], Henrique Kawakami[2], Ricardo Dahab[1]
(1) University of Campinas, Campinas, SP, Brazil
{gallo, dahab}@ic.unicamp.br
(2) KRYPTUS Information Security, Campinas, SP, Brazil
{gallo, kawakami}@kryptus.com

Abstract

Many high security applications rely ultimately on the security of hardware-based solutions in order to protect both data and code against tampering. For these applications, assuring the device's identity and integrity is paramount. In our work, we explore a number of factors that help to improve on device accreditation, by devising and defining both architectural and procedural requirements related to device construction, shipping and usage. Based on that, we proposed two integrity *shared verification schemes* which enable regular and auditing users of such applications to promptly and easily verify whether their interfacing hardware is trustworthy. We implemented our solutions in a key application, namely a hardware security module (HSM) suitable for use in supporting PKIs and also showed how it performs equally well in *Direct Recording Electronic* (DRE) voting machines.

2.1 Introduction

2.1.1 Motivation

Secure processors, such as those found in smartcards, and *hardware security modules* are key components of any production-grade PKI. With a minimal set of functionalities, these devices may offer an appropriate *root of trust*, from which a *trust chain* can be constructed, extending all the way to users' applications.

However, while secure processors and HSMs have been around for more than 20 years [2], little has been said about the necessary conditions for accreditation of a device's *root of trust* itself [15].

We believe that this gap poses a real threat to the goal of application- and enterprise-wide trust establishment, as any kind of *root of trust* compromise may render the entire system and its applications insecure.

2.1.2 Our Contribution

In this paper we explore a set of issues that have a strong influence on the level of device trust, encompassing conceptual, architectural and procedural aspects related to device construction, shipping and usage. For each of these, we propose refined and novel solutions drawn from our experience in developing and building a commercially available hardware security module designed for PKI use.

In order to provide the necessary background required for device trust establishment, we propose in this paper the concept of *cryptographic device identity* (CID), which extends the concept of a root of trust by adding stronger and more precise existence requirements than those found in other previous works. The CID and the proposed concept of *secure device epoch* (), are then used to construct *shared verification schemes* (s) with trust amplifying capabilities.

In this paper we also propose two such schemes, the Simple SVS and the (Byzantine) Traitor Evidencing SVS. We then implement the Simple SVS in a PKI-enabled HSM and show it performs equally well in such applications and in *Direct Recording Electronic* (DRE) voting machines.

2.1.3 Paper Organization

This paper is organized as follows: in Section 2.2 we describe the problem in greater detail and discuss some proposed solutions. In Section 2.3 we detail our proposals followed, in Section 2.4, by implementation results. Sections 2.5 and 2.6 conclude the paper, giving some ideas for future work.

2.2 The Problem and Related Work

Many security applications, such as PKIs, make use of hardware security modules in order to protect *critical security parameters* (s) from non-authorized use, disclosure, and modification, and to enforce usage control. The most common CSPs include key material in the form of (possibly many) secret and private keys.

However, CSPs are not restricted to cryptographic keys; for example, the key management algorithm implementations themselves must also be modification-protected, as tampering with them can lead to leakage of key material.

Another typical application that requires both data and code protection is *digital rights management* (DRM) of media contents. In this context, media (data) must be protected from both direct reading and subverted players (code). For this class of applications, specially at end user devices, trusted computing, TPM-based, solutions have been employed with reasonable, although sometimes limited, success, with many improvements being proposed [13, 73].

Threat models for DRM applications are peculiar. For example, whereas the leakage of a first-run movie can cause great losses to the producers, the adversaries' gains are much less obvious. For typical end user DRM applications, the gains an adversary can obtain are limited to its ability to redistribute and charge for the stolen media. The majority of TPM-based solutions targets the end user market and provides only limited protection. The weaknesses of TPM-based solutions stem from the fact that they are used both as passive root of trust and a single-chip, standalone device, not protecting main bus memory.

Only a few commercially available single-chip solutions, designed from scratch, such as the IBM Cell Processor [60], are strong against adversaries trying to abuse the *security API* and launch system-level physical bus attacks (non-invasive local attacks).

The Cerium secure computing architecture [11] enables certified (signed) program execution, protecting against physical and logical attacks by using a tamper-resistant CPU and a special secure micro kernel, along with a per-device secret key. The AEGIS single chip secure processor architecture proposal [63] is a notable advance, by providing a full encryption/authentication trusted computing base. The AEGIS proposal also provides, through Physical Unclonable Functions (s), a statistically unique quantity that can be used as the per-device unique key expected by Cerium.

For applications with higher attack rewards, the co-processor provided protection is naturally expected to be higher than single-chip solutions can provide [2, 5, 4]. The IBM 4758 [15] secure co-processor is a preeminent example of a device that poses a challenge to well funded adversaries. In order to protect against impersonation attacks, the 4758 is personalized with a private/public key pair at the factory, which developers can use to

identify a given device and protect their CSPs. A similar key pair was also latterly used by the Safekeyper [29].

Two key examples of high stake applications that may rely on device integrity for security are large government PKIs, (especially their CAs and RAs), and Direct Record Electronic (DRE) Voting Machines. Government PKIs tend to be very attractive targets for adversaries; for instance, in some countries, banking payment systems must use government issued digital certificates. In this context, strong subverting powers may act upon operational and auditing personal, posing a real threat to system reliability and, thus, their CSPs.

DRE voting machines are also under a strong, and quite interesting, adversary model. Protocol-based ingenious solutions have been proposed, including Neff's [44] and Chaum's [10], which try to avoid reliance on DRE physical and logical security by giving voters means to verify vote accountability. Despite these solutions, many attacks are still possible through DRE tampering [56].

Clearly, PKI and DRE have a common security requirement, namely platform/device accreditation. For these applications, very high levels of confidence on the device's integrity and identity are central in the effective protection of CSPs.

For the sake of the establishment of *trust* and *confidence* in a given device and, ultimately, on the entire application, security guarantees must focus on a number of issues, and not only on the device itself. These are: (i) system specification; (ii) system design; (iii) system implementation; (iv) device manufacturing; (v) in-factory device initialization; (vi) device shipping; (vii) device reception; (viii) device user setup; (ix) device use and operation; and (x) device disposal.

Ever since the early 1990s, concerns about trustworthy design and production of hardware security modules have been expressed, a fact reflected in the National Institute of Standards and Technology (NIST) publication FIPS 140 standard series [46]. The current version of this standard, FIPS PUB 140-2, covers in considerable relatively depth items (i) to (v) above, specifying clear requirements and recommendations for achieving device security. However, a large formal and methodological assurance gap exists between device shipping and user configuration, enabling severe security breaches such as hardware Trojans and device cloning.

There are other relevant standards that also apply to HSMs, such as the Common Criteria (ISO/IEC 15408) Protection Profiles (s). Some of the most common PPs are the Secure Signatured Creation Devices () (European standard CWA 14169) PP, and the BSI Cryptographic Modules Security Level Enhanced PP (-CC-PP-0036). Our proposed schemes are directly related to the Delivery and Operation (ADO) Security Assurance class of these PPs, as they allow the detection of modifications of the (i.e. the HSM) between the manufacturer's and the user's site, and also provide procedures for secure

installation, generation and start-up of the TOE.

2.2.1 Attacks: Device Cloning, Trojans, and Reverse Engineering

Although one can implement security controls for the design and manufacturing phases of an HSM, little can be done to ensure what happens after the device is shipped from the manufacturer without proper care.

Even a device with advanced tamper protection mechanisms can be easily defeated by replacing it with a clone device containing a hardware- or firmware-based Trojan, if no reliable identity and integrity tests can be conducted by the final user during ceremonial procedures.

Device cloning consists of building a device with the same physical appearance and features of the original device. In addition to the original features, the clone might incorporate malicious code/hardware, and must not possess effective tamper detection mechanisms in order to allow for the later extraction of sensitive information.

In our experience, typical rack mounted Ethernet HSMs, for example, could be cloned from an authentic device by replacing its internal electronics with a standard computing platform (e.g. an x86). For such devices, the only way for the end user to detect such attacks would be by inspecting security seals on the outside of the device's enclosure. Nevertheless, these seals [5] have a much lower security level than that expected from the device. Besides, the user will probably not be able to distinguish an authentic label from a fake one.

The clone device would probably be uncovered at some point in the device lifespan (due to firmware update procedures or maintenance), but by that time an adversary might already have performed an attack, and the system will be discredited.

2.2.2 Device Verification and PKI Operations

Even in critical PKIs, typical ceremonial operations surprisingly do not consider checking device identities. Figure 2.1 presents a real scenario, where concerns about the integrity of the CA HSM are not even mentioned.

The lack of ceremonial procedures to ensure the integrity of secure hardware modules is present at every level of the typical PKI hierarchy, from HSMs used by the Registration Authorities to those used by the root Certification Authority.

In addition to the tampered device threat, it is possible that one or more traitors exist and in collusion (or not) might try to subvert the applications root of trust. Obviously, not identifying those threats could lead to heavy and irreversible losses, especially in

1. The ceremony's participants are identified and invited to enter the safe room. The scripts steps are explained to every participant. Auditors and operators are introduced to the participants.
2. Certificate management servers and the HSM containing the CA key are powered on. Any basic setup is performed (e.g. clock adjustments)
3. The private key inside the HSM is reconstructed and enabled for usage through operators' smart-cards authorization.
4. CSRs are imported, verified and signed by the HSM using the CA private key. Newly signed certificates are exported.
5. HSM unloads the CA private key.
6. Backups are made and logs are generated for the HSM and for the certificate management servers.
7. Certificates are exhibited and published. The ceremony's minutes are printed and hand-signed by all participants.

Figura 2.1: Abridged script for certificate issuance by an offline CA drawn from the Brazilian National Academic PKI website [24] (ICP-EDU)

applications such as PKI and e-voting.

On occasions where a reliable third-party may not exist or is unavailable, deciding whether to trust the application is even more difficult. This may be the case when the deployment of private keys is necessary in emergency circumstances (e.g. an online delegated CA key disclosure, or encrypted military messages).

Election days are a typical situation where the three mentioned conditions may apply: there is no clear (reachable) trusted third-party, political party interests are in conflict, and DRE integrity may be in dispute. It is clear that practical tools are necessary to assist in diminishing the risks posed by device tampering and user betrayal.

2.3 Our Proposals

2.3.1 Trust Establishment Rationale

Establishing trust in a device has two dimensions: **procedural** and **computational**.

The first, which consists of **device accreditation**, is where auditing takes place, encompassing every stage related to the device's (and the application that makes use of it) life cycle, from system specification to system disposal.

Accreditation is heavily human bounded, as it is related to the confidence the user (or the enterprise) using a given application has on it. Ultimately, it can be expressed in *the trust one has in the device's root of trust*. In this dimension, trust (as security) is seen as process, that must be evaluated [25] from time to time, and that uses auditing as one if its key tools.

The computational dimension is built around the *accredited root of trust*, by cryptographic means, in the form of a *trust chain*. Usually, this chain is built through the use of asymmetric-key techniques, but not necessarily.

Binding the computational and procedural dimensions is thus fundamental for the trust establishment of a device and applications dependent on it, as HSMs and PKIs. In order to facilitate this binding, we propose and formalize the concept of device cryptographic identity - CID - and show how to use it to improve on a number of issues regarding hardware security modules, such as:

- assurance of the manufacturing process;
- securing the device's life cycle;
- prevention of important online and offline attacks.

2.3.2 Cryptographic Identity - CID

The cryptographic identity extends the trusted computing root of trust concept by attaching accreditation meaning and by imposing certain restrictive properties on it. Before continuing, we need to define the concepts of sealing and secure device epoch.

Definition: The sealing is the latest phase of the device manufacturing process where non-cryptographic verification (auditing) must be conducted, in order to decide whether a device can be accredited. The end of the sealing process defines the beginning of the secure device epoch (SDE).

Typically, secure hardware devices are manufactured either by the assembly of electronic components on a (multi-chip modules) or by the masking and encapsulation of silicon wafers (single-chip modules).

In either case, the security of the end device arises from a series of physical and logical features, which are embedded in the system during one or more phases of the manufacturing process.

The early manufacturing phases may allow the auditing of the system. During these phases, the firmware, components, or layout of the system can be freely scanned for integrity.

However, during a specific manufacturing phase, the device is physically or logically sealed, so that any further attempt to audit the system will be recognized as a tampering attempt, causing eventual destruction of keys and other CSPs within the device.

A secure co-processor or HSM, CID enabled, shall have a unique challengeable identity with the following properties:

- **Establishment**: the CID must be automatically created at the hardware's final (human or automated) physical inspection, at the start of the secure device epoch.
- **Uniqueness**: the CID must be statistically unique, per device, bound to a security parameter k.
- **Protection**: at device epoch, the CID shall be protected inside a cryptographic boundary, with physical protections so that: (i) it must be improbable that an adversary could clone or copy a CID from a device without breaking through the device's cryptographic boundary, (ii) the CID must be destroyed with high probability whenever any attempt to violate the device's cryptographic boundary is made.
- **Verification**: the CID must be verifiable with negligible statistical "false positive"probability (key-size dependent)

- **Integrity**: device operation, and related CSPs protected by the secure device, should be possible if and only if the CID is preserved.

2.3.3 Considerations about Building a CID-enabled Device

In a typical tamper-responsive HSM, there are at least two conceptually distinct components enclosed inside the cryptographic boundary: the Sensing Unit () and the Processing Unit ().

The PU has more processing power and less storage constrains. This unit effectively implements the various cryptographic primitives, key storage, key management, and other functions. Although its persistent memory cannot be easily erased, its contents can be encrypted with a key stored in the SU non-volatile .

The SU is capable of storing sensitive information in a non-volatile memory, and destroying it upon detection of a tamper attempt.

In a multi-chip module, the SU may consist of a low-power microcontroller with built-in or external sensors, and SRAM (Static RAM) or (Ferroelectric RAM) to store sensitive information. A single-chip module may contain an on-chip, low-power, non-volatile SRAM. It may also contain tamper sensors, and also external inputs pins that directly trigger SRAM wiping.

Typically, the SU is battery powered when there is no available external power source, in order to provide the continuous monitoring of the device's cryptographic boundaries, and the storage of sensitive information.

In the described scenario, the device CID can consist of keys generated after the device enters the secure device epoch, using a secure random number generator as entropy source, and stored inside the SU or PU.

In the latter case, it would be also necessary to protect the CID keys with a master key stored in the SU, allowing for easy CID destruction in case of a tamper attempt. As an added bonus, this master key can also be used to encrypt and to authenticate CSPs and firmware inside the cryptographic perimeter.

The solution that comprises a master key stored in the SU and a larger asymmetric key pair stored in the PU is usually preferable, as multi-chip SU modules may have very limited processing and storage capacity. In single-chip modules, the SU may not have any processing power at all.

It would also be possible to build the CID upon the evaluation of PUFs, generating keys that encrypt sensitive, non-volatile information. A single-chip scheme was presented by the AEGIS architecture proposal [63], and multi-chip schemes would also be possible, although there are no known commercial products using this technology.

2.3.4 Verifying the Device's CID

The properties of a CID-enabled device allow strong device integrity and trust accreditation. Accreditation is done by auditing means at the beginning of the *secure device epoch*, satisfying the **establishment** property. That is the last instant in which specialized auditing personnel would be required to unambiguously attest a hardware security module's identity.

From that moment on, only logical (cryptographic) challenges are required to attest a device's integrity, by the **verification** property. This is true because any attempt to gain unauthorized access to the CID material would destroy the device's identity as required by the **protection** property.

Generally speaking, challenging the device can be achieved by using a composition of classical cryptographic primitives. The most immediate approach is to have a user client to send a nonce to be signed by the hardware security module using a key protected by the trust chain rooted at the CID.

At first, verifying the nonce signature would be enough to attest the CID enabled device's integrity and identity. However, this is not as simple as it seems. In fact, this verification process *only moves* the security and trust requirements from the verified entity () to the verifier entity ().

Therefore, if the verifier entity is not trusted at a trust level similar to that expected from the verified entity, we have, in fact, degradation of the global trust level.

To solve this problem we propose *shared verification schemes* which, by using multiple (possibly less trustworthy) VREs, *amplify* the total verifier trust level, possibly beyond that required for the VDE.

All verification schemes are comprised by a set of common items:

i) N, a set of trust weighted nodes, that encompass VREs and the VDE;

ii) T, a set of trust directed edges between nodes in N, that express the trust relationships among nodes, forming a graph G with set of nodes N, and set of edges T;

iii) M, a trust metric that evaluates G and a security parameter P used to decide whether the VDE is trustworthy [34].

It is also assumed that the verifier device presents the result of the verification process directly to the HSM user (e.g. by means of a visual indicator). In other words, the verifier shall not rely on a communication channel that is controlled by the device that is being verified.

In the following, let E_p be the probability that the VDE has been tampered with in a given time frame, in such a way that the attacker has gained total control over it *without* destroying the respective CID; i.e., a **CID-preserving attack**. Note that the more protections a device employs, the closest E_p is to zero, as there is a high chance that the tamper attempt will be detected by a tamper detection mechanism. E_p is also different from the probability of the VDE being tampered with but losing the original CID (a CID non-preserving attack), due to the activity of tamper response mechanisms. The clone attack is an example of a CID non-preserving attack. We call this second probability E_d, and assume it to be significantly greater than zero.

Although E_d is typically much larger than E_p, CID non-preserving attacks will have the same efficacy of CID preserving attacks in the cases where the HSM end user cannot check the authenticity of the CID. As attackers follow the path of least resistance, CID non-preserving attacks would thus be the chosen ones, and all the efforts of physical tamper detection would be foiled. Thus, our objective is to thwart these **CID non-preserving attacks** through detection, bringing E_d as close to E_p (and consequently close to zero) as possible. In this way, HSM tamper detection and reaction mechanisms can really deliver the security level that they were designed to.

Simple SVS (): our simple shared verification scheme employs multiple, independent verifiers that, in consensus, must vouch for VDE accreditation.

Let P_i be the probability that the i^{th} VRE has been tampered with. Let n be number of distinct verifiers. Then, if our trust metrics accepts a single CID identity challenge mismatch as a signal that the verified identity is not authentic, then E gives us the VRE-set *composed tampering probability*:

$E = \prod_{i=1}^{n} P_i$, where $P_i \in (0..1)$.

Although simple, this expression is elucidating, as it tells us that tampering with the shared verification can be made very difficult, if not negligible, just by increasing the number of VREs. That is true because an attacker would need to tamper simultaneously with all the VREs in order to produce a fake result.

Of course, this expression assumes that compromise probabilities are independent, reinforcing the need for the special procedures for VREs production and deployment that will be described in Section 2.4. These procedures reduce the probability of simultaneously cloning all devices (a "parallel clone"attack) to a level much lower than that of a standard clone attack, even if all VREs share the same hardware design. The S-SVS successfully amplifies verifiers' trust level even with very simple assumptions; this will be especially useful in the DRE voting machine context. Figure 2.2- "S-SVS Graph"presents the simple

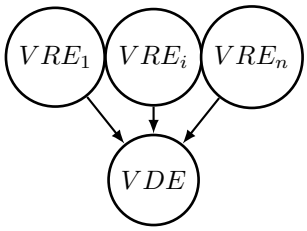


Figura 2.2: S-SVS Graph

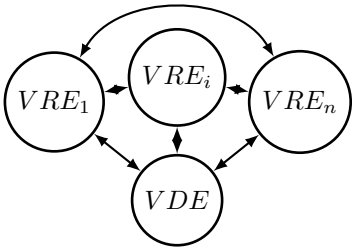


Figura 2.3: TE-SVS Graph

scheme, with G an n-degree directed tree.

While simple and powerful, the chosen trust metrics also makes it possible for an attacker, a traitor (a legitimate user that acts as an attacker), to cause denial of service attacks, by denying a single authentic signature.

Traitor Evidencing SVS (): our traitor-evidencing shared verification scheme employs n multiple, cross-verified, verifiers that in consensus either vouch for VDE accreditation or point up to t possible traitors. The key idea is that every VRE and the VDE itself cross-checks every other device, creating a complete graph G, with $n(n-1)$ edges. Thus, each device would challenge and be challenged by every other device. The implementation of such scheme could be done using special USB OTG(On-The-Go) tokens that connect to each other by a special powering hub.

The problem of identifying traitors on the TE-SVS is similar to the classical Byzantine Generals problem [31] as our model assumes no reliable third-party beyond the VDE and the VREs and only direct communication between "generals". As we have n "generals"deciding whether to accredit or not the VDE, it follows that it can evidence up to t traitors, where t is given by $n \geq 3t + 1$. For larger values of t, it is impossible to decide who is a traitor.

The TE-SVS graph has a clear advantage over the S-SVS one, as it could be made robust against treason by allowing VDE accreditation with a least $n - t$ vouching VREs. However, the composed tampering probability function would suffer a small degradation because tampering with $n - t$ VREs would suffice for a successful attack.

For the sake of simplicity, we will assume for TE-SVS the same trust metric as for S-SVS, even though the TE-SVS graph now allows for much more robust metrics such as group trust [33]. Fortunately, this scheme can promptly make evident a traitor as its identity is checked by every other device on the graph; the possibility of being caught is now the main inhibitor to treason.

2.4 SVS Architectures, Implementation, and Results

Any SVS architecture must provide components for easy trust verification in a measurable fashion, composing the elements required by the shared verification scheme.

Essentially, the CID verification challenge is the action that builds single edges on the trust graph G. The trust representation, however, do not pose any strong restriction on the nature of the challenge behind the edge. Thus, there is more than one way to prove device integrity by using the CID-rooted trust chain.

2.4.1 S-SVS Architecture for Cryptographically-Enabled VREs (Cryptographic Tokens)

The most direct form of CID challenge is through the use of an asymmetric cryptographic signing scheme, with a public key pk, a private key sk, a statistically unique *nonce*, a signing function $signature = sf(key, data)$, and a verification function $response = vf(key, data, signature)$, where $response \in \{checkok, checkfailed\}$. A given VRE chooses randomly a nonce, sends it to the previously known challenged entity, which applies sf to it, and returns the signature which is checked by using vf .

The main advantage of this method is that the verification primitives can easily be made computationally intractable, by using suitable mechanisms and appropriate key sizes in relatively constrained devices, as cryptographic tokens.

The system requirements for the VDE and the VRE are not unusual: (i) both must have a (possible unsafe) local communication channel, and (ii) they must have a human interface device that exports the security API function vf.

The VRE and VDE production, the verification procedure, and the interstice between them are described as follows:

VRE and VDE Production: Before the secure device epoch starts, a parameter n is chosen by the device manufacturer, depending on the final application of the secure device. Parameter n defines the maximum number of VREs to be used to verify a given VDE. As the idea is not to depend on any reliable third-party (external trust source, as a CA), VDEs and VREs are *paired* at device production.

During the manufacturing process, the HSM and each of the tokens generate a key pair. The HSM public key is stored inside each token, and all token public keys are stored inside the HSM. The keys inside the HSM are protected against modifications by the CID key material, so they will became invalid if any tamper attempt is detected. The HSM and tokens are then logically sealed (entering the SDE), and no more key exports can be made, nor regeneration of key pairs.

In order to maximize the independence of compromise probabilities for the VREs, thus minimizing the E probability, some basic rules shall be enforced:

Interstice: After manufacture, the two device sets (VDE and VREs) follow distinct logistical paths, and are stored at distinct physical locations. They will only meet again at the end user facilities. The auditing tokens can be kept inside a vault or in possession of mutually conflicting interests.

VDE Ceremonial Verification: The ceremonial-time device verification can be easily achieved. It requires that $1 \leq j \leq n$ randomly chosen auditors, each possessing a VRE, connect their devices to the VDE. For each verification device, one cryptographic challenge is launched by the device itself and other is initiated by the VDE. The challenge response is then output through the human interface device (*hid*) as a single bit message. The VDE challenge is used to indicate whether a given VRE has been replaced, maliciously or not.

2.4.2 Token Based Implementation

The token-based architecture was employed to implement a variant of the Simple Scheme (S-SVS) in our PKI-enabled HSM. This variant adds the authentication of the VRE (token) by the VDE (HSM), allowing the detection of an invalid VRE.

A special token was developed using an microcontroller platform and (Elliptic-curve Cryptography) primitives. Besides the , this device contains a status that presents the result of the verification process directly to the HSM user. This effectively exports the security to the end user. The HSM presents the result of the VRE authentication on its built-in display.

The HSM booting process was also modified so that it now only allows user initialization after the successful authentication of a minimum number of VRE tokens, during a CID verification procedure by the user. This not only avoids unauthorized HSM initialization, but also enforces CID verification by the user. The aforementioned verification procedures were applied to the CA ceremonial script presented in 2.1, which resulted in the updated step 2 in 2.4.

2.1 Certificate management servers and HSM containing the CA key are powered on.

2.2 VDE integrity verification is performed by successive VRE interactions

2.3 Any basic setup is performed (e.g. clock adjustments)

Figura 2.4: Update step 2 of CA script presented in figure 2.1

One of the benefits of this solution is that the security risks associated with storage and shipment procedures are greatly diminished. The developed VREs could be manufactured for very low prices, as their required computational power and tamper-resistance are minimal. As each VRE is inexpensive, the added cost of the USB token solution is quite low, even for a relatively large number of devices.

2.4.3 SVS Architecture for Highly Constrained VREs

For severely cost-restricted verifier entities, the token-based solution presented may be unsuitable. That may be the case in elections, where a large number (hundreds of thousands) of DREs must have their identity checked by an even larger number (millions) of candidates' representatives. In these cases, the computational power of the VREs is restricted to be very low or even near to null.

However, in spite of these strong restrictions, the trust edges of the trust graph G can still be constructed, as long as the verified entity provides specific functionalities, namely: (i) an output human interface device (), and (ii) a high stability secure real time clock.

The output hid has no security requirements; it may only allow a user to acquire information from the VDE without further devices. The high stability secure real time clock () is an extended version of the secure clock required found in [69]. Further than the requirements that it shall be (i) non-resettable, and (ii) monotonic, we demand that it shall be (iii) highly accurate, and (iv) linear with respect of time.

In this context, we explored the VREs and VDEs asymmetry allowed by the simple trust graph to construct a Time-Based One Time Verification Code (), similar to existing Time-Based One Time Password [41, 42]. Like the One-Time Password, the security of the TOTV relies on the security of a symmetric key. The VREs and VDE production, the verification procedure, and the interstice between them are described as follows:

VREs and VDE Production: Before the secure device epoch, four parameters (n, s, t, d) are chosen by the device manufacturer, depending on the final application of the secure device. Parameter n defines the maximum number of VREs paired to this VDE,

t delimits the time frame in which the VDE will be verifiable (usually related to the device's operational life-time), s defines a time step, i.e. the minimum time between two consecutives checks, and d the number of digits. Moreover, at the SDE, at the secure manufacturing location, after the CID is generated, n verification symmetric keys are randomly chosen by the VDE and protected through encryption by the CID key material. Then, for each of these keys, $\frac{t}{s}$ OTP values are output by the VDE, forming an independent verification sheet (), totalizing $n \times \frac{t}{s} \times d$ OTP digits. Each IVS is then isolated and reserved for later delivery and use by the ceremonial auditing personnel.

Interstice: during the interstice, each IVS and the VDE must be kept away, preferably under conflicting interest control, so that an adversary, possibly a traitor, cannot gain access to more than one IVS. The IVSs must be stored safely to prevent forward attacks. The VDE has no further security requirements, as any attempt to attack the VDE device would destroy the device's trust chain, and, thus the TOTV symmetric keys.

VDE Ceremonial Verification: The ceremonial-time device verification is straightforward. It requires only very little memory (to store d-character OTP values) and only enough power from the VRE to perfome comparisons, thus very suitable for human beings. If we use the simple trust metrics, each auditing user memorizes (or copies) from her IVS one or two expected OTP values prior to the device's deployment. Then, at ceremonial time, the VDE device outputs through its hid the calculated OTP values; the VRE sole action is to compare these values, trusting or not the device. If every VRE vouches positively, the device is accredited.

2.4.4 TOTV Implementation

We implemented the TOTV solution in our PKI-enabled HSM. In our implementation, the VDE hid was the same two line, twenty digit LCD display that was already available on the hardware secure device we built.

In our HSM design, the requirement for a HSSRTC was translated as a temperature-compensated, battery-backed RTC enclosed inside the cryptographic perimeter, yielding a tamper-resistant clock. The employed HSSRTC has a 5 ppm time drift, not deviating more than 2 minutes per year.

The TOTV algorithm used was similar to the algorithm for user authentication in web services proposed by the initiative [41, 42]; the sole differences are concentrated on how we use the produced OTP values. In our implementation, we consider the human user as the verifier of the OTP values produced by the secure co-processor (in contrast to the authentication server). The prover, instead of being a token with a single symmetric

key, is the HSM itself, that holds not one symmetric key, but n distinct symmetric keys protected by the trust chain rooted at the CID.

In our implementation, oriented to HSMs, we used values 5, $\frac{1}{2}$ day, 10 year, 6 digits as n, s, t, and d parameters, yielding 5 IVS with 43K char each. The 43K chars were printed on A4 paper with font size 8, resulting in a four page IVS, perfectly suitable for almost any ceremonial use.

The generation of the OTP value was done by the 533MHz x86 main processor of the HSM by using the SHA256 based OTH [41], in negligible time. It is important to note that even low processing power CID-enabled devices would generate the IVS quite quickly.

For the parameters we chose, typical ceremonies require only a 12 character memory, as the 10 year accumulated deviation is only 20 minutes, significantly smaller than step s.

As we could see, the use of the TOTV verification is very convenient as it is very cheap to implement for multiple VREs and enables off-line use, typical of root CA operational environments and massive elections.

General-Purpose HSM Considerations: Clearly both the token based and the TOTV SVS implementations can be employed by general-purpose HSMs successfully. However, for certain applications such as SSL acceleration, the additional VRE ceremonial verification steps, if not automated, may represent some operational burden.

Voting Machine Considerations: For CID-enabled DRE voting machines, the use of the TOTV solution would not dramatically change the parameters presented by the HSM implementation. As long as simple modifications are made, typical IVS will have $1 \sim 2$K chars. The trick is to print only the IVS of the electoral days, as they are usually pre-defined by law.

Suppose that for large countries we have at most 20 political parties; then checks would be made at 30 minutes intervals, the DRE would be used for 10 electoral days (one election per year with 10 hour duration, over ten years), and a 6 char OTP. For that setup we will have 20 IVS with some 1200 char.

2.5 Conclusion

In this paper we analyzed the importance of device identity establishment and verification in PKI and voting applications.

We presented two specific solutions that enable regular and auditing users to promptly and easily verify whether their interfacing hardware is trustworthy with high levels of

confidence at ceremonial time, as the employment of multiple verification devices greatly increases the required effort to clone or subvert the hardware.

The first solution is token based, in which we implemented very low priced verification devices to be used in amplified trust verification schemes.

The second solution relies on a Time-Based One Time Verification Code, and can be used at extremely cost-sensitive applications, but it requires more control over the VREs in the interstice phase and a secure clock inside the VDE. It was shown how this solution could be easily implemented in HSMs and DRE voting machines. We also presented details of our implementation of the first solution on our hardware security module (HSM), and how it prevents unauthorized HSM initialization and diminishes the security risks present during equipment shipment and storage.

2.6 Future Work

"If you cannot measure, you cannot improve- Lord Kelvin's quote. Our future work is concentrated in developing a *trust metric framework* which encompasses device trust, ranging from device production, device epoch to device disposal, allowing for better integration with other analytical metric tools already applied to entire applications or enterprises.

We are also working on how to better estimate the E_d and E_p tamper probabilities in real world devices.

Acknowledgements

We would like to thank the referees for their valuable comments.

Capítulo 3

T-DRE: A Hardware Trusted Computing Base for Direct Recording Electronic Vote Machines

Publicação: Este artigo foi apresentado no *"26th Annual Computer Security Applications Conference - ACSAC2010"*, na data de 5 a 9 de Dezembro de 2010, em Austin, Texas, EUA e publicado pela ACM nos *proceedings* do evento.

Roberto Gallo[1,2], Henrique Kawakami[2], Ricardo Dahab[1],
Rafael Azevedo[3], Saulo Lima[3], Guido Araújo[1]
(1) University of Campinas, Campinas, SP, Brazil
{gallo, dahab, guido}@ic.unicamp.br
(2) KRYPTUS Cryptographic Engineering Ltd., Campinas, SP, Brazil
{gallo, kawakami}@kryptus.com
(3) Tribunal Superior Eleitoral, Brasilia, DF, Brazil
{rafael, saulo}@tse.gov.br

Abstract

We present a hardware trusted computing base () aimed at Direct Recording Voting Machines (T-DRE), with novel design features concerning vote privacy, device verifiability, signed-code execution and device resilience. Our proposal is largely compliant with the (Voluntary Voting System Guidelines), while also strengthening some of its recommendations. To the best of our knowledge, T-DRE is the first architecture to employ multi-level, certification-based, hardware-enforced privileges to the running software. T-DRE also makes a solid case for the feasibility of strong security systems: it is the basis of 165,000 voting machines, set to be used in a large upcoming national election. In short,

our contribution is a viable computational trusted base for both modern and classical voting protocols.

3.1 Introduction

Electronic voting systems (s) are a very interesting subject, as they are comprised of system components which interact within an complex environment with boundary conditions of different nature, legal, cultural, logistical and financial. Several countries have adopted EVSs, tailoring them to meet their specificities.

The Brazilian voting system currently has over 135 million registered voters [8], with variable literacy degree. Thus, electronic voting is a very simple procedure, which consists of typing candidates' numbers on a reduced keyboard, guided by simple instructions on a small screen. Brazil adopted Direct Recording Electronic voting machines (DREs from now on) in 1996. In 2009 a decision was made to replace part of the aging hardware base with a newly designed version, while maintaining backward compatibility.

Voting Systems Fundamental Goals

In spite of local constraints, EVSs share six common, fundamental, goals (Sastry [56]):

Goal 1. One voter/one vote. The cast ballots should exactly represent the votes cast by legitimate voters. Malicious parties should not be able to add, duplicate, or delete ballots.

Goal 2. Cast-as-intended. Voters should be able to reliably and easily cast the ballots that they intend to cast.

Goal 3. Counted-as-cast. The final tally should be an accurate count of the ballots that have been cast.

Goal 4. Verifiability. It should be possible for participants in the voting process to prove that the voting system obeys certain properties.

Goal 5. Privacy. Ballots and certain events during the voting process should remain secret.

Goal 6. Coercion resistance. A voter should not be able to prove how she voted, to a third party not present in the voting booth.

These goals are related (e.g. a voting system that does not satisfy goal 5 will hardly satisfy goal 6) and potentially conflicting (e.g. it is not trivial to build a voting system that is

totally verifiable while preserving voters' privacy). Third-party end-to-end verifiability has been a recurrent subject [52]. Usually, verifiability is linked to the concept of (statistical) confidence level. Different cultures, and thus electoral laws, have different thresholds for the level of confidence they consider adequate for the electoral process.

Software independence is not enough. Different voting protocols [10, 44, 12] have been proposed to meet the above goals, with variable degrees of success and effectiveness. Unfortunately, most of them can be defeated by compromised software or hardware running in the underlying computing base. In order to mitigate such threats, software-independent systems were proposed by Rivest and Wack [53]: *A voting system is* software-independent () *if an undetected change or error in its software cannot cause an undetectable change or error in an election outcome.* However strong, this concept ensures most of the above requirements but not all.

For instance, coercion resistance and vote privacy are especially susceptible to attacks based on tampered hardware and software, as vote input devices themselves can leak information [28, 54, 56]. Hardware protection and verification is thus an essential aspect, regardless of whether SI systems are employed or not. While some effort has been done towards the specification of hardware functionalities in order to provide sufficient device accreditation and tamper resistance [47, 17, 56], there is much room for improvement on the path to feasible implementations. Here we follow that path, presenting a hardware trusted computing base (TCB) for direct recording electronic voting architecture, T-DRE in short, suitable for a variety of existing voting protocols and systems.

Summary of our Contributions

Our contributions are present both in the novelty of the T-DRE components and in their composition. Namely, we propose a trusted hardware architecture that extensively employs signed code execution with hardware-enforced access control to peripherals in order to prevent a number of attacks. Further advancements include human-computable device integrity verification mechanisms, strong accountability, and improved signed-code execution assurance, all supported by a certification hierarchy which takes advantage of the proposed hardware.

The T-DRE architecture described herein was adopted by the Brazilian National Election Authority (Tribunal Superior Eleitoral - TSE). In order to fully validate the specification, we first implemented a prototype evaluation platform. Subsequently, the specification was realized by a vendor under TSE's control, using another hardware platform, and taking into account additional costs and stringent field, legal, and resilience restrictions, while maintaining backward compatibility with the deployed base. This endeavor, which resulted in 165,000 produced units, further supports our claims on the feasibility of the architecture.

Our proposal is not an airtight solution to electronic voting; we discuss its limitations in Section 3.5. However, we do claim that it provides a layer of security to SI and non-SI systems alike, whose strength is degrees above that of voting systems currently deployed around the world, by making it extremely difficult and costly for a fraud attempt to go undetected. Also, although we target centralized elections, in Section 3.4.2 we discuss how T-DRE can be naturally extended to decentralized environments such as in the USA.

This paper is organized as follows: Section 3.2 gives practical goals and boundary conditions of voting systems; Section 3.3 discusses related work; Section 3.4 details our proposal; Section 3.5 reports implementation efforts; Section 3.6 concludes, with ideas for future work.

3.2 Voting Systems Practical Goals and Boundary Conditions

Attaining the fundamental goals is subject to practical boundary conditions, especially in large elections. Three important constraints are:

Availability. Voting systems must be available during the critical periods (election day, tallying, etc.) and resist denial of service attempts. DRE machines must resist tampering;

Credibility. An aspect of utmost importance, it is at the basis of fair representativity. Accordingly, implementations of voting systems should minimize the chance of operational errors and resist tampering. Here, again, DRE hardware security and verifiability plays an important role;

Resource Rationalization. The practical realization of voting systems should take into account various cost-related variables, such as auditing and hardware cost and maintenance. When security is considered, a clear budget trade-off exists between built-in security mechanisms and the security procedures employed by the Electoral Authority (). While the first is typically a one-time expenditure which is multiplied by the number of DRE machines, the second is recurrent, flexible, and proportional to the number of polls. The security targets for DRE machines must take this into account.

Security Targets

The specification of security targets should make provisions for many different variables (Common Criteria [65]). In face of the current Brazilian Electoral Laws, the following

variables demand special attention:

Window of opportunity. Our implementation should take into account that attacks on DRE machines can occur at any time, but more easily in the interstices between elections. Pre-election time is the most vulnerable due to transportation of DRE machines across huge distances.

Surface and scope of attacks. Voting machines are subject to different levels of adversarial exposure between procedural checkpoints established by the EA: during election interstice, an adversary can have physical access to the DREs; in the pre-election (setup) phase, adversaries may have media (logical) access to the DREs; at election day, adversaries typically have only operational access to DREs, as all non-HID I/O are sealed and the machines operate offline. Our security target take these conditions into account. It provides tampering resistance and tampering evidence on the Critical Security Parameters (CSP) such as keys and key counters, with a physical security target of FIPS 140-2 level 3 [46] (passive resistance). Moreover, a successful attack must have limited scope - breaking one DRE should not increase the chances of an adversary of breaking another.

Level of adversarial expertise. Attacks on a DRE, especially those which adulterate or recover key material or CSPs, must demand multiple experts, considerable time (impossible to execute during election day) and removal to a laboratory with special equipment.

Audit control points, mechanisms and equipments. Audit points shall be precise, clear and accessible. There should be an audit point aggregator that simply expresses the DRE's state (fully operational, in error, in service). The interpretation of this audit point should not require additional equipment nor complex procedures, being accessible to all parties involved in the electoral process: voter, electoral authority, poll worker, and party advocates.

3.3 Related Work

In this section we discuss related work regarding T-DRE's features.

3.3.1 Signed Code Execution

Signed code execution [11, 2] is an important tool in voting systems [55, 67]. Many security issues faced by EVSs can be directly mitigated by the proper use of signed code execution. Benefits include:

- ensuring that only official voting software is executed in DREs, enhancing resilience against deliberate adulteration and operational errors which may violate EVS fundamental goals such as vote secrecy and coercion resistance;
- tracing and accountability of incidents, enabling security through legal means;
- simple verification of binaries' integrity in pre, intra and post-election phases, which facilitates auditing by parties, voters, and the Electoral Authority.

Hardware-based signed code execution can be achieved by various means, the *de facto* standard being the Trusted Computing Group (TCG, now ISO/IEC 11889) Personal Computer Trusted Platform Module (TPM) [26], a companion chip to the main system CPU, usually connected via LPC bus. The TPM has functional characteristics similar to a smart card. In cryptographic terms, the TPM performs several operations: key generation, storage and use of cryptographic keys, protected by a key that represents the system's root of trust. Moreover, unlike typical smart cards, the TPM has mechanisms for software attestation, which allows certain running application parameters to be anonymously verified and certified as not tampered. The module is recommended by the VVSG ([67], Section 5.5.1) for protection of the DRE software stack.

One of the drawbacks of PC TPM modules is that they work passively, in hardware terms, with respect to the main system CPU. TPMs, by design, can be completely bypassed by the system's boot sequence if the BIOS (especifically, the "Core Root of Trust for Measurement", CRTM) is tampered with, and thus "deceived" when used in application verification tests. Extensions to the TPM as the TEM from Costan et al [13], being also passive with respect to the CPU, represent no improvement in this regard.

To overcome this master-slave problem, one can consider the sole use of secure processors as the main component of a TCB aimed at DREs. However, even state-of-the-art processors with security features, such as AEGIS [63], USIP-PRO [38] and Cell [60], suffer from impeditive shortcomings. While the AEGIS specification is completely open, to the best of our knowledge there are no commercially available realizations of it. The USIP-PRO, in turn, has limited processing power, its architecture is proprietary and the vendor makes no assertions regarding memory protection against data modification. Finally, the Cell processor is proprietary, not allowing full access to hardware features from independent software vendors, thus adding undesirable obscurity to the design.

3.3.2 Key Management and Certification

Entertainment platforms have guided the industry regarding the execution of signed code for DRM purposes. Microsoft's Xbox [22] and Sony Playstation 3 execute only code signed by keys directly under vendors' root CAs. With the Cell Processor [60], Sony advances

further: unsigned code running on PS3 has limited access to the device's peripherals, notably the GPU. Only signed code has full access to hardware features. The VVSG (Section 5.5.1) forbids non-signed code from running on DRE hardware, similarly to console platforms. The VVSG also recommends a TPM-like component for controlling software execution.

In addition to certifying (signing) the voting machine software stack, cryptographic key material is extensively used in many voting systems [67, 55, 10, 45, 44] for other reasons, from voting, to producing closeout records, audit log signature and verification, to encryption/decryption of votes and other sensitive material.

Although key management and storage could be handled in software by the DRE, cryptographic tamper-resistant hardware is preferred. The VVSG recommends the existence of a hardware tamper-proof signature module () in DREs, whose primary function is to manage the life cycle of two asymmetric key pairs: i) the Election Signature Key (), a unique per-election/per-device key used to sign votes and closeout records; and b) a per-device DRE Signature Key (), which identifies the device and is used to produce certificates for the ESK. The usage of DSK and ESK is strictly controlled by the SM by means of two counters: CountESK and CountDSK. CountDSK counts the number of generated ESK certificates ever signed by DSK. CountESK counts the number of ESK usages. When the closeout record is produced, ESK is erased by the MSM and both counters are included in the resulting record.

3.3.3 DRE System Verification

Easy auditing is a paramount requirement for voting systems as it is central to the establishment of trust on the DREs' integrity and correct operation. The concerns with integrity verification of the entire DRE system stack (hardware, firmware, and software) are not new. Although auxiliary devices (software or hardware) can be used, ideally solutions should provide effective user-computable verification mechanisms of the DRE integrity, so that less, not more, hardware and software components are used to verify the main system. In this sense, device integrity verification itself should be also software-independent.

Sastry [56] describes a handful of desired DRE verifying properties, mainly aiming at software insulation, by constructing a proof-of-concept DRE with multiple (seven) processors. Gennaro et al [21] establish a condition for tamper-proofness of general hardware and give some clues on how to check device integrity by means of cryptographic challenges. Öksüzoglu and Wallach [47] present, in VoteBox Nano, an elegant human-verifiable software and firmware (FPGA bitstreams) checking mechanism based on random "session identifiers", which change every time the DRE is rebooted. Gallo et al [17] generalize Gen-

naro et al's conditions, prototyping a human-readable, cryptographically-strong system verification method called Time-Base One-Time Verification (TOTV), which allows for multiple device verification in a trust amplifying fashion, making humans part of the verification protocol. Although both [47, 17] can be used by poll workers and party advocates to assert DRE integrity, they are not practical for large-scale verification by voters, as they require comparison of multiple digit verification numbers, a hindrance when illiterate voters are considered.

3.4 Our Proposals

3.4.1 The T-DRE Architecture and the Master Security Module

The T-DRE architecture was devised to meet security and availability requirements, as well as cost restrictions. Some key requirements are:

- **(R1)** Run solely signed code, even if the opponent has operational access to the DRE media.
- **(R2)** Enforce the verification of the entire software stack, from the BIOS to the voting application, establishing an effective software trust chain;
- **(R3)** Allow the system state (integrity) to be widely attested by any user. Voters, party advocates and the electoral authority (EA) should be able to verify the integrity of the DRE without additional electronic devices;
- **(R4)** Resist physical and logical attacks, preventing unauthorized access to key material and application tampering;
- **(R5)** Contain only fully auditable components, enabling thorough system verification by the EA and the society;
- **(R6)** Allow the use of low cost, widely available hardware components, with reasonable computing power and fully open source development chain;
- **(R7)** Allow maintenance of the DRE machine and upgrade of its cryptographic mechanisms during its long expected lifetime (10 years);
- **(R8)** Enable and ease software and firmware development cycle, including field testing and simulations; allow faithful simulations which are clearly verifiable as such, which includes the production of non-valid results only.

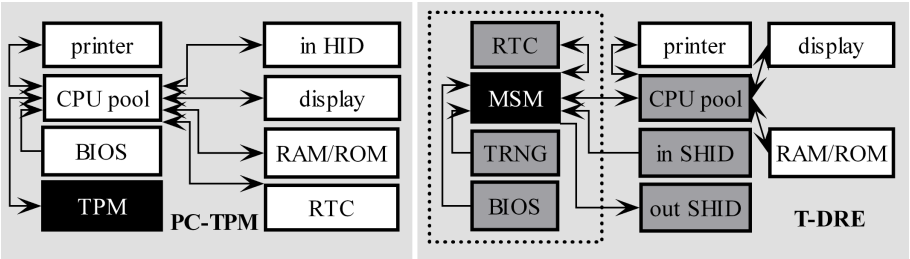


Figura 3.1: PC-TPM architecture (left) and the T-DRE architecture. The T-DRE components surrounded by the dotted box are under physical protection; BIOS physical protection is optional. Dark-gray components are under MSM direct control.

In order to achieve these objectives, we based our proposal on the fundamentals of secure hardware presented by Gennaro et al [21] and Gallo et al [17]. The latter introduces the concept of *cryptographic identity*, which states conditions for the establishment and verification of a root of trust for general secure hardware. Both suggest the use of their verification schemes in DREs. Here we go further, presenting a DRE system architecture which also brings new control mechanisms and a new verification method (Section 3.4.3).

Our architecture is depicted in Figure 3.1, along with a classical PC-TPM system. In both, the CPU pool (one or more main processors) is the main processing unit, which runs the voting application (and software stack). In the PC-TPM design, the CPU pool is the bus master of all peripherals, including the TPM chip, which can be completely bypassed by tampered software at boot time. There is no way for the TPM to prevent CPU access to peripherals, nor to inform users that non-signed code is running.

The T-DRE Architecture, in contrast, is fundamentally different from the PC-TPM: the security is based on the proposed Master Security Module (), which concentrates the DRE's cryptographic mechanisms and controls system peripherals (encrypted voter keypad, poll worker terminal, status lights), BIOS, and CPU pool. This centralization allows for a multi-level certification-based peripherals' access policy which can be enforced on the software running on the CPU pool. This is further explained in Section 3.4.2. The MSM control over the human interface devices (HID) also plays crucial role in our solution. Its implications are explored in Section 3.4.3. The MSM is also a CID-enabled device, i.e. a device whose root of trust, represented by a cryptographic key, is bound to the device's physical integrity: crossing the cryptographic boundary is highly likely to cause the device's root key destruction (and thus its identity), preventing the production of valid closeout voting records.

The T-DRE Software Verification, in contrast to PC-TPM, allows for full software stack verification, including BIOS. Prior to the CPU boot, after the DRE hardware power-up, the MSM checks the authenticity (and possibly decrypts) the BIOS contents; only

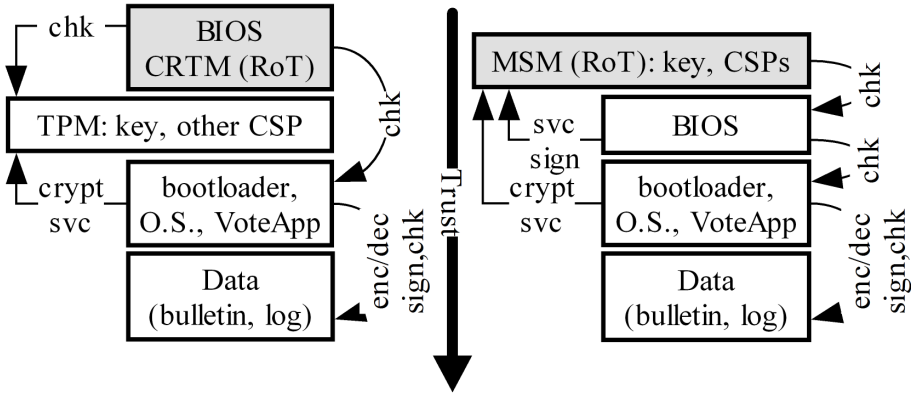


Figura 3.2: Verification chain for code execution, PC TPM and our proposed MSM

if a valid (signed) BIOS is found, the CPU pool is able to boot. Now the CPU runs signed code from the very beginning of the boot sequence and is able to use the MSM to check the remaining of the software stack (bootloader, O.S., voting applications, scripts, configuration data). The differences between the T-DRE and the PC-TPM boot processes are illustrated in Figure 3.2. It goes beyond VVSG's required signed code verifier hardware module (VVSG, Section 5.5.1).

Both the T-DRE peripheral architecture and the software verification mechanisms are novel to DREs. Moreover, the MSM also acts as a VVSG Signature Module (VVSG Section 5.1.2). In spite of these advancements, our architecture can be implemented with off-the-shelf electronic components, enabling secure, fully auditable systems and low cost realizations. In Section 3.5 we describe a prototype using only commodity, general purpose components.

3.4.2 Hardware-Reinforced Certification-Based Privileges

Satisfying Section 3.1 goals (in special privacy) and Section 3.4.1 requisites (in special R3, 5, 7, and 8) requires strict control over the DRE software. Only official (highly audited) voting software must be able to produce valid closeout records. Maintenance (loosely audited) software must be prevented from accessing the DRE's key material (thus preventing production of valid closeout records) and from running an apparently valid, but otherwise fake poll (thus breaking privacy). Also, voting software being developed must be able to exercise all DRE features without being able to produce valid tallies or deceiving voters.

To attain the desired software control, we combined the MSM's control over the DRE's peripherals and the running software stack, with a custom key hierarchy based on Public Key Infrastructure (PKI) technology (with established procedures and audit controls),

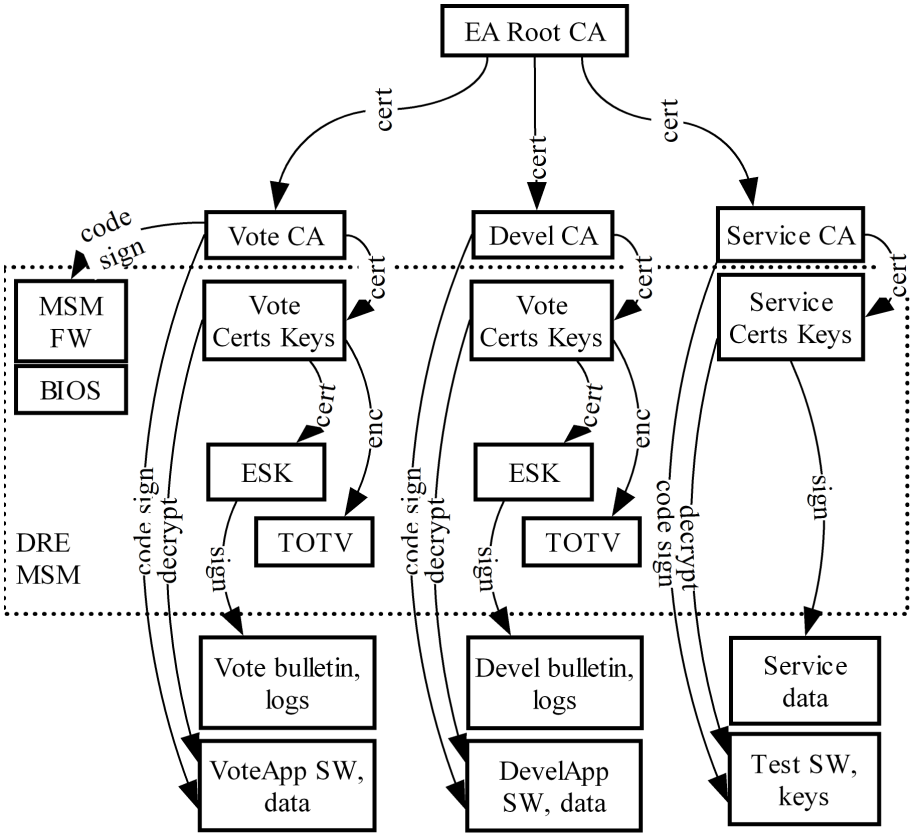


Figura 3.3: Certification hierarchy, code and data, and key usage

thus reducing required audit points. Our proposal centers the confidence of the electoral system on the EA root certification authority (EA-rootCA), which is audited (cryptographically) by the parties and the society. Figure 3.3 illustrates the PKI architecture with its three intermediate CAs, VoteCA, DevelCA, and ServiceCA, each with distinct purposes and privileges. In common, these CAs are responsible for": a) managing the DSK certificate life cycle; b) signing the DRE's software stack; and c) decrypting any messages coming from the DRE, when the voting protocol so demands. Software signed under each certification branch has different execution privileges and access to different key materials. Each DRE has three DSK certificates (and key pairs), one for each tree branch. *All DRE certificates (and corresponding keys) are stored within the MSM, which controls both the key usage and the signed code execution privileges.*

Vote CA Branch: Binaries signed under this branch have total control over the DRE hardware and are used in the actual election days - they have access to the official voting key material (DSK_{vote}, ESK_{vote}), producing valid election closeout records, controlling the voter's keypad use, the poll worker's keypad use, and the access to the Secure Output HID (Section 3.4.3). The MSM is responsible for enforcing the privileges of the signed

CA Privilegies	VoteCA	DevelCA	ServiceCA
Key Material	$(DSK)_{vote}$ $(ESK)_{vote}$ $(Others)_{vote}$	$(DSK)_{devel}$ $(ESK)_{devel}$ $(Others)_{devel}$	$DSK_{service}$
Input HID access	Full	Full	Restricted
Output HID access	Full	Full	Only test results
Security API (Secure HID)	Full	Restricted	None

Tabela 3.1: Signed code execution privileges for our DRE proposal; MSM enforcement

code over the DRE hardware, without any software interference.

Development CA Branch enables the necessary functions for development and election simulation activities , granting restricted access to peripherals and keys: i) the MSM produces signatures only with DSK_{devel}, ESK_{devel}, $Other_{devel}$ keys; and ii) the signed code has no access to the secure output HID which signals valid polls. This prevents in-development code from being used to deceive voters, and easily distinguishes valid signatures on real closeout records from those produced under simulation.

Service CA Branch enables DRE maintenance (memory, battery, peripherals testing and systems components replacement). Servicing operations are highly distributed, thus hard to audit. Under ServiceCA, signed code is not allowed access to keypads nor the secure output HID nor any ESK key material. The allowed operations are: a) repairing the input/output of cryptographic devices, and b) signatures of maintenance logs. Table 3.1 summarizes the privileges enforced by the MSM in each certification branch.

Other Considerations: Although our proposal targets centralized elections, it can be naturally extended to decentralized scenarios, as those in the USA, by adding Local Electoral Authorities CAs (as additional intermediate CAs) to the tree of Figure 3.3. Then, each local authority would maintain three CAs (VoteCA_{local}, DevelCA_{local}, ServiceCA_{local}). This allows a great deal of independence and flexibility, where local authorities can produce and run their own software without depending on the national authority. Furthermore, DREs can be easily shared by local authorities.

3.4.3 T-DRE Verification: Secure Human Interface - S-HI

Integrity verification schemes provide variable confidence level in their output. As a rule, the better the scheme the more intrusive an adversary has to be in order to fake a result.

From less to more intrusive we list: software modification (), hardware modification (), and key extraction from hardware (). Human verification is especially hard to attain if tampering with the communication channel between the user and the system under verification is a possibility. *We call a human interface secure (S-HI) up to a class of intervention (, ,) if it does not produce false results even when it is subject to tampering of that class.*

The VoteBox Nano random number display (along with its verification scheme) is S-HI-SWM, i.e., it resists logical (bitstream) attacks, but not S-HI-HWM. In T-DRE we provide users with two interfaces: one S-HI-SWM and one S-HI-HWM. For the S-HI-SWM interface, we employ the MSM (hardware-)controlled 'out ' (Figure 3.1) as a four-state LED which indicates VoteCA, DevelCA, ServiceCA, and non/corrupted signed code. This is a clear improvement over VoteBox nano, as we attain the same security level with a much simpler user verification scheme.

For the S-HI-HWM interface, we employ a modified version of TOTV [17] that does not require the high-stability secure real-time clock (HSSRTC) of Gallo et al's solution. The TOTV protocol is similar to the Time-Base One-Time password (TOTP) described in [42]; TOTP derives, from time to time, an n-digit sequence from a secret key known to the *verified* device and possibly to the *verifier*. It is defined as $TOTP = HOTP(K, T)$ where T represents the number of time steps between the initial counter time T_0 and the current Unix time. K is a key, and $HTOP$ is the HMAC-based One-Time Password Algorithm defined (RFC 4226 [41]) as $HOTP(K, C) = Trunc(HMAC - SHA - 1(K, C))$. The TOTV proposal binds the secret derivation key K to the device's cryptographic identity (CID), so that any attempt to tamper with the device, by construction, should destroy the CID and thus cease the TOTV sequence creation. In our architecture, we maintain two TOTV keys (K_{vote}, K_{devel}) protected by DSK_{vote} and DSK_{devel} keys.

In order to check the integrity of a specific DRE, a user has to access a TOTV sequence produced by the electoral authority. In other to avoid replay attacks, this access must be either i) confidential and prior to the DRE display of the TOTV, or ii) real-time, on-demand, and signed.

In our proposal, we use the same construction as the TOTV, but instead of having a single T representing the number of time steps since Unix epoch, we use two T variables (T_{vote}, T_{devel}). These represent the time steps accumulated during every DRE usage when running in voting mode and development mode, respectively. The time counters necessary for this are made persistent and are protected by the MSM from stalls or decrement. In order to avoid other types of replay attacks, and after signed closeout records are produced by the DRE, it stalls the counter and includes it in the certificate, pausing the timing increments. In the next DRE usage (possibly on the next election), the electoral authority sends the poll workers $TOTV = HTOP(K, T)$, with $T = max(T_{closeout}, T_{user-access})$,

which allows for DRE boot-up and counter resumption. The modification from the original TOTV proposal is motivated by the cost of a high stability secure real time clock. The usage of our proposals is further illustrated in Section 3.5.

3.5 T-DRE Implementation & Results

The practical realization of our proposals was done in two phases, a prototyping and a mass production phase. In the first, the theoretical, technological, and procedural solutions were tested and validated. In the second, any necessary modifications were implemented.

3.5.1 Hardware and Firmware Implementation

Prototype Due to the large number of DREs to be produced (165,000), our proposals were thoroughly tested in a prototype prior to the delivery of final specifications to the chosen vendor for mass production. In the prototype (composed by two connected boards: B1 and B2), we instantiated all of the T-DRE main peripherals (Figure 3.1, namely: MSM, BIOS memory, encrypted voter keyboard (in SHID), output device (serial display), secure output (out SHID), main CPU, among others. The B2 board is a commercial embedded PC, with an AMD Geode LX800 CPU, with 256MB RAM. The B1 is a custom board specifically built for the prototype. It hosts the MSM and other devices, and connects the security module to the bottom board by means of an ISP connection (to BIOS delivery) and a USB connection (for other, cryptographic, services).

Considerable effort was spent on the correct choice of the micro-controller (uC) employed for the MSM as it must conform to many requirements: a) have internal code and data memory (both persistent and volatile); b) the entire memory must be lockable (no read/write access); c) memories must be large enough to handle cryptographic mechanisms (RSA, ECDH, ECDSA, SHA-2, homomorphic DH) and store keys and certificates; and d) reasonable performance, in order to handle quick BIOS verification and cryptographic services.

In our prototype, the MSM was implemented using a NXP LCP2000 (ARM) family uC which meets these requirements: a) up to 1MB internal FLASH memory with code read protection, b) up to 40KB RAM, enough for the implementation of asymmetric algorithms; c) 72MHz, 32-bit core, with 64 DMIPS performance. The voter input device (cryptographic, tamper-resistant physical keyboard) was simulated using a MSP430 uC, connected to the main uC by an bus. The output secure HID is composed by three light emitting diodes (LEDs) which are directly connected to the MSM. In order to provide an onboard source of entropy, we implemented two random number generators using

avalanche-effect semiconductor noise.

For the asymmetric algorithms on the MSM and the cryptographic keyboard we used the RELIC library [14]. For our prototype, the implementation of the required MSM functionalities, including DSK and ESK handling, binary code verification, CSR exportation, secure firmware update and cryptographic keyboard handling required about 180Kbyte FLASH (code) memory and 24Kbyte RAM. Employed functions were: signing and verification, asymmetric encryption/decryption (RSA-2048 PKCS#1); hash (FIPS 180-3 SHA-512); block ciphers (FIPS 197 AES 256).

A prototype software stack was also implemented. The bottom board BIOS was modified so that it uses the MSM slave interface to check the bootloader's authenticity. The bootloader was also modified (from) to test the boot image, rather than files, using the MSM.

Attacks and Countermeasures

T-DRE, as PC-TPM, has no effective runtime (after boot) countermeasures against defective software nor buffer overflow attacks (data execution). While the first problem can be traced (and later dealt with) due to the sole use of signed code, the second demands more attention. In Brazil DREs have no data links, so buffer overflow attacks from voters or poll workers keypad is highly unlikely. For further protection, one may consider the "reboot prior to each vote" approach.

Hardware systems are subject to many implementation attacks, in special side-channel analysis (SCA) [28]. use information leaked through side-channels from real systems. More information can be found in [28] and [54]. SCA-aware cryptographic hardware usually resists, to a certain extent, side-channel attacks. However, they typically suffer from lack of transparency on the employed security mechanisms (see Section 3.3). As we privilege transparency over off-the-shelf solutions, our solution uses a standard uC and added FIPS 140-2 level 3 equivalent physical protection and SCA counter measures:

- The entire top board was immersed in tamper-resistant and -evidencing resin;
- In order to weaken power attacks (, ,), we adopted two countermeasures: a) we used decoupling elements in all external communication paths; and b) we filtered and stabilized the power input to prevent energy consumption variation;
- Timing attacks are weakened by using constant-time cryptographic operations.

Mass Production Versions

After validation, our architecture was realized in a mass production version, and is set to be used on the 2010 Brazilian national election, with more than 165,000 DREs. This

version differs from our prototype in some implementation decisions and functions: a) there is a single board containing all the components required in our architecture; b) the CPU pool was implemented as a single x86 processor; c) the MSM master interface was replaced by an assistive (supervisor) interface; if the MSM perceives any BIOS change, it resets the CPU pool (the main drawback being that BIOS cannot be encrypted). A second mass production version is expected to be manufactured in the fourth quarter of 2010, with more than 200,000 DREs. These will present further side-channel countermeasures and incorporate improvements deemed necessary.

3.5.2 Usage Procedures

Pre-Election, Election, and Post-Election Procedures

Since valid (non-tampered) voting machines run only code signed by the electoral authority, it is easy for a verifier to check whether the voting application is correct and that the voting machines have not been tampered with:

- In the **pre-election** phase, a human verifier must: a) Check for any physical tamper evidences on the DRE; if any are found, stop and report; b) switch on the DRE and enter the "resume TOTV" provided by the electoral authority (Section 3.4.3); if the DRE fails to continue the boot process, stop (either it is not the correct DRE or the device has been tampered with); c) check for the next TOTV to be shown by the DRE; if it is not the expected one, stop (the DRE has been tampered with); d) perform other verification procedures (e.g. audit procedures).
- On **election day**, human verifiers can, at any time: a) check for software stack integrity, by simply checking a DRE's status S-HID (indicative LED); if the S-HID does not present a valid status, the use of that DRE must be prevented (either it has been tampered with or it is not running the correct voting software stack); b) from time-to-time, electoral judges and voters can check for device integrity by comparing the TOTV produced by the DRE with those from the electoral authority; if any comparison fails, stop that DRE's use (it has been tampered with).
- In the **post-election** phase, a human verifier must check whether the final TOTV present in the closeout record is valid; if not, the device has been tampered with and the produced closeout record is deemed invalid.

Other Procedures: Development, Testing, and Maintenance

We chose a PKI model for key management, so that its established practices and procedures can be used. The use of the root CA's and the VoteCA' authorization keys is only

granted to the highest rank staff of the EA (in Brazil, Supreme Court judges preside the Supreme Electoral Court), audited (cryptographically) by political parties, Congress and society representatives.

3.6 Conclusion and Future Work

In this paper we propose T-DRE, a trusted computing base for direct recording electronic voting machines, which is mostly independent of the voting application and largely VVSG-compliant. T-DRE's novel combination of technologies enable device verifiability by humans, deep PKI integration and simple auditing. Our architecture was prototyped and then reengineered for large scale manufacturing, with 165,000 devices produced. These DREs will be used in the Brazilian 2010 presidential election.

T-DRE's main component, the Master Security Module (MSM), unifies the TPM and SM modules proposed in the VVSG and adds key new features by: a) enforcing, over the entire software stack, a policy of multi-level, certificate-based access to peripherals and key material; and b) taking control of human interface devices, thus amplifying vote privacy and user DRE tamper detection.

We also indicate how the new audit and control mechanisms present in our architecture can be integrated into the usual electoral cycle, the voting itself, election simulation, device testing and servicing, and software development.

Currently, we are working on the design of a fully-auditable secure processor to be used as a CPU-MSM for DREs.

Capítulo 4

FORTUNA - A Probabilistic Framework for Early Design Stages of Hardware-Based Secure Systems

Publicação: Este artigo foi apresentado no *"5th International Conference on Network and System Security"*, na data de 6 a 8 de Setembro de 2011, em Milão, Itália, evento co-patrocinado e com publicação pela IEEE.

Roberto Gallo[1], Henrique Kawakami[2], Ricardo Dahab[1]
(1) Campinas State University, Campinas, SP, Brazil
{gallo, rdahab}@ic.unicamp.br
(2) KRYPTUS, Campinas, SP, Brazil
{gallo, kawakami}@kryptus.com

Abstract

This paper introduces FORTUNA, a probabilistic framework that supports the conception and early design stages of hardware-based secure systems. FORTUNA can point out potential weaknesses of complex systems, involving physical and logical attacks, basic human interaction or even a few classes of unknown threats. FORTUNA consists of two main elements: a) a logical-probabilistic theoretic model in which quantitative and qualitative security assessments of hardware-based systems can be done; and b) a semi-automatic tool, based on the proposed model, that can assist secure system designing from the very initial development stages. To the best of our knowledge, FORTUNA is the first framework (and tool) to support such a broad scope of interactions and also the first aimed at the conception and early design phases of hardware-based systems. Other

contributions include a proof of the "policy of least privileges" under our model and an example of use of the framework in the design of a secure microprocessor.

4.1 Introduction

4.1.1 Motivation

The security of real and complex systems, composed of hardware and software interacting with people, represents a fundamental and important goal, hitherto surprisingly difficult to achieve and easily tampered with. Examples of these systems include: HSMs, DRE voting machines, tokens, and cryptographic processors. We observe that system security breaches are a statistical certainty: all we can do is to estimate when, how and how much effort is needed for the system to be broken.

In our view, this is due mainly to the multidisciplinary nature of security, involving areas such as number theory, digital design, physics, statistics, human and legal factors, trust, and secure programming techniques, among others. To date, there is no unified security theory, a problem which is only made worse when one remembers that opponents always look for the weakest link, each opponent within his/her own specialty.

This multifaceted nature of the problem greatly hinders the conception, design and implementation of secure systems. It is thus crucial to bring about a holistic view of security, even without a unifying theory. Without such a view, hardware systems are currently designed based on a series of heuristics, guidelines, standards and tools with narrow scope, often relying on a (group) designer with vast experience and able to balance risks, countermeasures and system costs empirically.

With FORTUNA we aim at changing the conception and design of hardware-based secure systems into a less heuristic process, automating the cost and security assessments in the very early stages of system design, minimizing rework and the total .

4.1.2 Our Contributions

FORTUNA is unique in several aspects. To the best of our knowledge, it is the only framework that supports the design of secure hardware-based systems in a broader setting, covering aspects from human interaction to physical attacks.

FORTUNA is also the design support tool that can be applied earliest in the process of secure device development, which allows for overall cost reduction and appreciable increase in systemic security.

From a theoretical viewpoint, with FORTUNA's logical-probabilistic model we were able to prove the well known golden rule of "minimum access policy", while challenging

other high level policies as "minimizing the trusted computing base".

This paper also brings a brief example of the use of FORTUNA in the conception and design of a real secure processor.

4.1.3 Paper Organization

In Section 4.2 we discuss the problem of hardware-based secure device development in further detail. Section 4.3 presents our conceptual model and related theoretical results. In Section 4.4 we describe the implementation of our framework and its application in the development of a real secure processor, giving practical results. Section 4.5 describes related work. In Section 4.6 we conclude and suggest future work possibilities.

4.2 The problem in greater detail

The importance of real implementations of secure systems based on both hardware and software is fully established. However, despite its importance, the attainment of secure systems has proven to be surprisingly difficult, even when ample resources are available at all stages: design, implementation, auditing, testing, etc.

As mentioned above, the reasons seem to be varied and numerous: the area of security is very interdisciplinary, a unified security theory is nonexistent, and the fact that an attacker needs (and will) to aim only at the weakest links. But these are not the only reasons. An important part of the problem is due to design issues. When a designer starts to account for the many design aspects (e.g. system architecture, assumptions, evaluations, attack scenarios), the task rapidly becomes intractable, even when only high-level aspects of security are considered.

One symptom that illustrates the relevance of the problem is the change that the security community faces today, from asking "is this system secure?" to "for how long will this system (or solution) remain secure?". Ultimately, this change expresses the reality of the physical world and the fact that every implementation of a real system has a nontrivial stochastic nature. The odds of security flaws are embedded in the very nature of the systems, either in the physical components or in the incompleteness of logical descriptions. Ignoring the probabilistic nature of the problem is a serious oversight.

When we consider that systems in general have security as an expected characteristic rather than an objective, and these systems' objectives are subject to the "cost vs. performance vs. features"race paradigm, it is clear that the design of economically viable secure systems with time-to-market constraints is a very complex task. It is both a scientific and a technological challenge.

Defining security under these terms is also crucial. We adopt the economic view: an attack is successful (viable) when the opponent's reward is greater than the cost of the attack ($G_{adv} > C_{adv}$). These values, however, can only be determined after the attack; thus, before carrying it, the opponent needs to deal with the **expected** costs and rewards eC_{adv}, eG_{adv}. On the other hand, a system is viable when the the costs for protecting it are smaller than the expected losses in the case of security breaches ($C_{prot} < eL_{fail}$). Again, probabilities are present[1].

To deal with this complexity, researchers and designers have employed a myriad of methods with different scopes, efficiency and depth. In terms of scope, the most comprehensive are the meta-standards such as the Common Criteria (CC) [65], policies and risk assessment methodologies. A little more specific are standards as NIST's FIPS 140-2 [46] and CC Protection Profiles which set system requirements and dictate the expected system behaviors.

As an illustration of two high-level recommendations, we mentioned the "policy of least privilege" which has appeared in the literature for over 20 years. The policy of least privilege recommends that a system principal (user, process, component) must have the least possible access to resources (keys, data, applications...) to perform its functions. A variation of this crucial policy will be proved with our model. A more recent policy is the "minimization of the trusted computing base". This policy states that systems must minimize the amount (code, area, resources) consumed by trusted services in order to minimize possible security flaws.

With narrowest scope but deeper reach, we find novel algorithms, secure implementation techniques, and even system emulation tools [48]. We emphasize the existence of automatic formal security provers which use formal logical systems descriptions (not physical). The underlying problem, however are shown to be either NP-hard or intractable [23, 27, 51]. There are also works that employ fully heuristic methods for security evaluation, for instance, with the use of virtual machines [49] (of common system architectures or small variations). For a more general use, there are risk assessment methodologies typically focused on software and network security [57, 35]. Also, it is worth mentioning that, to the best of our knowledge, there are no tools that deal with both logical and physical aspects of secure systems. Further, there is yet no framework, supported by a comprehensive model, to assist in the stages of conception and early development of secure systems based on hardware and which allows for quick architectural adjustments

[1]It is also interesting to note that the relationship between attacks and systems viability is marginal because the correlation between eL_{fail} and G_{adv} may be very weak when the considered system handles non-financial logic. As an example, take the DRE (direct recording electronic voting machines) case. How to measure gains (when a fraud succeeds) and losses (of people's will)? Another example is the case of DRM. As pointed out in [17], the gain for an adversary when copying a pre-release movie is very different from the potential losses of a studio.

while considering logical, physical and cost aspects.

4.3 Our Proposals

Our model is based on a set of observations and derived properties, rooted in very basic system characteristics:

- Obs1: a secure system can be composed of other systems (or components)[2].
- Obs2: the security of systems has a probabilistic nature.
- Obs3: individually insecure (with respect to a given policy) components can be arranged in the form of a secure system[3].
- Obs4: secure components (with respect to a given policy) may be arranged into an insecure system.
- Obs5: ultimately, all components are physical. Logical components are abstractions represented in a particular physical component configuration (or state)[4][5].
- Obs6: There are no complete descriptions of non-trivial practical systems[6].
- Obs7: Every component has an associated cost for its deployment (e_{comp}).
- Obs8: Certain (typically local) components are associated with adversary rewards ($g_{comp,adv}$)[7].

Based on the above observations we extracted five properties to be used as the bases for our model:

- B1: **interaction channel**: every subsystem that can be composed with others has one interaction channel. This interaction channel may be a logical abstraction, providing a communication channel. The channel can be directed or not. Interaction

[2]System and component examples are: processor, memory, metal case, cryptographic key, user, and algorithm implementation.

[3]Security policies can be composed.

[4]Logical components are abstractions of physical properties where only a fraction of the available entropy is considered and the rest is ignored.

[5]The exploit of the ignored information present at the physical level allowed the discovery of side channel attacks [28].

[6]Both from a physical and practical perspective: at design-time, knowledge is partial and resources limited.

[7]e.g. user data, cryptographic keys, authentication data.

channels can be classified hierarchically, regarding available description levels. The channel can be as generic as "mechanically connected" or specific as " connection". Components can have multiple connections (even between a given pair).

- B2: **entropic potential**: represents the information assets that generate benefits for the opponent. The benefits can be directly useful (e.g. a user's private key) or indirect (e.g. a key that wraps the private user key). Entropic potential is directly associated to Critical Security Parameters (CSPs) [46]. Measured in bits.

- B3: **entropic impedance (or resistance to leakage)**: quantifies the permeability of components and interaction channels to entropy. It measures how well the information present in a component is made available and is incorporated into others around it through interaction channels. It is given as the probability that a given entropy amount migrates in a given timeframe from a component A to a component B trough an interaction channel AB.

- B4: **implicit security**: components, with a certain security policy set are subject to different attacks. Each attack can affect one or more of the security policies. Each attack has a different cost and different success probability. Thus, one can associate to a given policy a pair (p, c). Intuitively, $c = f(p)$ or, in other words, the more resource one puts in the attack, the greater the odds of succeeding.

- B5: **security provided**: expresses the ability an (directional) interaction has of transporting the implicit security experienced by a component A to a component B. Together with the implicit security, it expresses the "protection relationship" (A protects B) or, in other words, how the security policy implemented by A affects B through an interaction channel.

These five properties describe many of the features of high-level architectures of secure systems, especially at the conception and design phases. They apply to both physical and logical systems and can be used to model systems at the earliest development phases. *Our goal is to build models for security assessment based on the architecture of the system under development, not vice versa. This allows the system architect to focus efforts on the design of the solution, using his/her standard development tools.* This way, the developer can implement the changes suggested by FORTUNA quickly and easily. To this end, we performed two types of modeling: the first uses graphs (Section 4.3.1), mainly for illustrative purposes, and the second uses ProbLog, a logical-probabilistic language (Section 4.3.4).

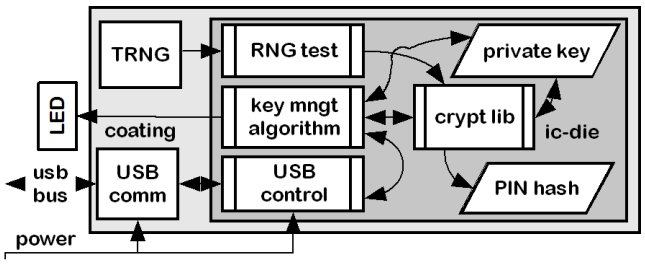


Figura 4.1: Graphical representation of a simple cryptographic token. The outer box represents the protective coating. The inner box, the IC die (microcontroller core) with logical components inside.

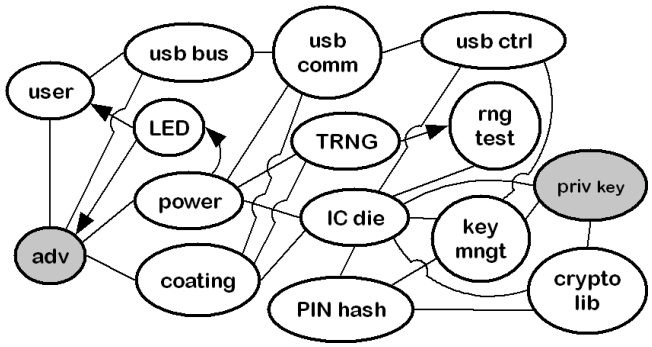


Figura 4.2: Graph representation of the same token architecture but obtained applying observations 1 to 8 and properties B1, B2 and B3.

4.3.1 Graph Modeling

Figure 4.1 shows the simplified architecture of a standard cryptographic token, a simple secure system. The diagram brings information of both physical and logical nature. Rectangular items indicate physical components, trapezoids indicate data items, and double-walled rectangles represent procedures (firmware components). Arrows indicate explicit interaction channels. Channels can be either logical or physical.

Figure 4.2 shows the same architecture as figure 4.1 but with a graph representation, and adding user and adversary roles. All components are mapped to vertices. All interaction channels are mapped to edges (or arcs) between the components. The "contains" relation between the container and the contained components are mapped by connecting all contained items to the container (e.g. coating and TRNG, IC die and USB comm.).

Using properties B1 to B3, and marking "priv key" as the adversary's asset of interest, we can trace the ways in which this asset can leak to the opponent. An analogy that facilitates understanding is to resistive circuits (Kirchhoff's Laws): the private key represents the VCC and the opponent the GND. All edges are resistive. Vertices represent nodes. We are interested in an architecture that minimizes the entropic flow (current) "priv key"

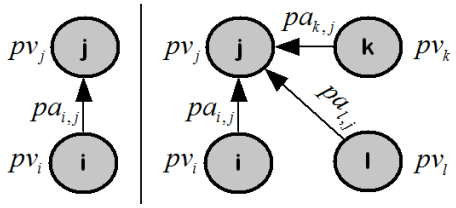


Figura 4.3: Definition 1: vertices and relations.

toward the opponent. Obviously, the analogy is limited since we do not have "the sum of current zero" but, instead, "the entropy of a channel is less than or equal to the sum of all entries".

Figure 4.2 facilitates qualitative security analysis. For example, it is easy to see that the logical architecture of the token does not maximize security: multiple edges incident to component "priv key"increase the likelihood of information leakage – there are multiple paths by which information can leak from "priv key" to the opponent. The same graphical model allows for immediate intervention: removing link between "key mngt alg"and "priv key"reduces risks. This modification conforms to the "least privileges policy". Formalizing all this, we have

Definition 1 (Model 1): Let $D = (V, A)$ be a digraph representing a related system and external agents that interact with it. V is the set of vertices representing the system constituents and external stakeholders (users, opponents). Let A be the set of arcs representing the interaction channel (B1). Let s be a bit of the secret S of interest to the opponent (B2) and which the system protects. By properties B1 and B3, we observe that s leaks from its container through the arcs with probability $pa_{i,j}$ (the probability that s transits from vertex i to j through arc ij), so that vertex j now has probability pv_j of knowing s. We are interested in minimizing pv for the vertex that represents the attacker.

In the following we consider all probabilistic events independent. Given Definition 1, pv_j in figure 4.3 is given by:

$$pv_j = pa_{i,j} \times pv_i, \tag{4.1}$$

where $pa_{i,j}$ is given. When j has multiple edges or incident arcs (Figure 4.3), it suffices that s leaks through only one arc; so, for each vertex j we can write:

$$pv_j = 1 - \prod_{i \in N_D^-(j)} (1 - pv_i \times pa_{i,j}), \tag{4.2}$$

where $N_D^-(j)$ is the set of all in-neighbors of j. For all n vertices of D we can write the same equality with a maximum of n variables in pv_i, which provides us with a system

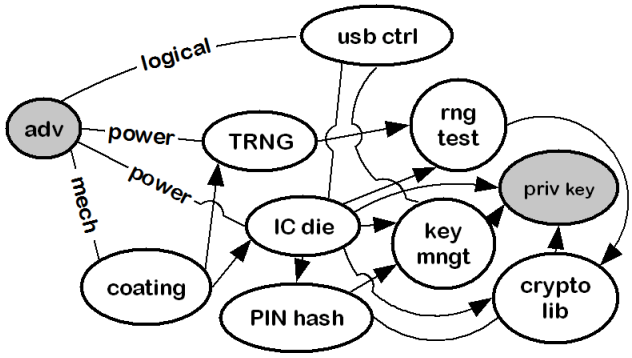


Figura 4.4: Protection (or security depends) relationship. Graph representation of the same token architecture but obtained applying observations 1 to 8 and properties B1, B4 and B5.

of degree n totally determined. For the vertex k that contains s, pv_k is trivially 1.

Figure 4.4 shows the same architecture of Figure 4.1. This digraph is a re-orientation of a subgraph of Figure 4.2, when applying properties B1, B4, and B5. This digraph represents the "protection" relationship and can be read alternatively as "the safety of vertex j depends on vertex i". We observe that the protection relations begin at the physical components and end at the logical ones, respecting the natural concept of "contained" - logical components are contained in one (or more) physical component(s). Interestingly, the protection relation has wider scope when compared with the entropy graph, as it can model different types of security objetives (e.g. integrity, availability), not only confidentiality. The representation of Figure 4.4 is related to Schneier's attack trees [59].

The graph of Figure 4.4 can be either split into a set of graphs, each for a security objective, or annotated so that it correctly captures security dependencies. In either case, it may require designer intervention to qualify the relations, especially regarding logical conditions (and, or)[8]. Obviously, for large systems, the required annotation amount may be impeditive and hinder the benefits of the graph analysis. For this reason, in Section 4.3.4 we present an alternative based on DTProbLog, a probabilistic derivation of the Prolog language. Now, we formalize:

Definition 2: Let $D = (V, A)$ be a connected digraph representing part of a system. V is the set of vertices representing components that establish relations of protection. By Obs8, for each vertex v of V there is a cost e_v (financial, work, performance) associated with the use of v. Let A be the set of arcs representing the protection relationships (or security dependence) via a given interaction channel. By B4, for each arc uv of A there is a cost c_{uv} associated with a given probability of success pp_{uv}. The arcs incident on v can be composed in and/or form. Let C be a subset of V representing the system's CSP.

[8]e. g. the security of component A depends on component B (and/or) C.

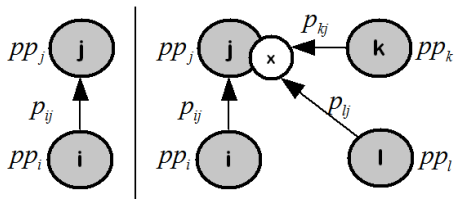


Figura 4.5: Definition 2: vertices and relations.

By Obs9, to each vertex v of C is associated a gain g_v. We are interested in making the best attack plan more expensive than the expected gain for the adversary.

In the following we consider all probabilistic events independent. Given Definition 2, the expected probability of success of an attack on vertex j of Figure 4.5 is given by $pp_j = p_{ij}$, while the expected cost of a successful attack on j via arc ij is given by:

$$e_j = e_{ij} = \frac{f(p_{ij})}{p_{ij}} + e_i, \tag{4.3}$$

where p_{ij} and $f(p_{ij})$ are given and e_i is the cost of attacking vertex i. When multiple protection relations are established, the cost is given by the smallest attack value among all possibilities (incident arcs). The problem is therefore related to the "shortest-path problem", but with some changes: when an "and" relationship is established ($\times$ sign in the figure) all "shortest paths" must be accounted. We can write the relationship for each vertex in the graph as:

$$e_j = min\left(\sum_{x\in X} e_{xj}, \sum_{y\in Y} e_{yj}, ..., \sum_{z\in Z} e_{zj}\right), \tag{4.4}$$

where X, Y and Z represent sets of arcs grouped around the "and" notation and $N_D^-(j) = \bigcup X \bigcup Y \bigcup, ... \bigcup Z$ and $\oslash = X \bigcap Y, \vee X, Y$. Now, the probability that a given vertex is successfully attacked is given by:

$$pp_j = 1 - \prod_{X\in N_D^-}\left(1 - \prod_{i\in X} p_{ij}\right). \tag{4.5}$$

4.3.2 Graph Model Analysis and Results

The two models obtained from Observation 1 to 8 and Properties 1 to 5 allow not only for automatic security evaluation and design insights but it also permits theoretical exploration. In this section we prove two well-accepted security policies under our model.

Policy 1: *Grant to system principals the least privileges necessary to perform their jobs.* Explanation: this policy states that, in order to improve security, system principals, such as users and processes, must be granted access only to the resources needed for the task. That means restrictions on accessed data (user, system), cryptographic keys, processing and storage resources.

Theorem 1: Policy 1 either does not affect, or it improves the overall system security regarding confidentiality CSPs.

Proof: We use "Definition 1". The principals are represented by vertices that relate to (access) confidential resources (vertices that contain the s bit) via one or more arcs (either directly or chained). We want to minimize pv_j of equation 4.2 for the vertex that represents the adversary. Thus, it suffices to show that any arc uv removal never make pv_j larger. We have two cases based on whether the arc uv is present in the equation for vertex j or not. If it is, an arc removal implies that the product of equation 4.2 has one less iteration. Since each term $1 - pv_i \times pa_{i,j}$ is always equal to or smaller than 1, this removal either does not change the total value of the product or increases it. That means that the value for pv_j in equation 4.2 either diminishes or does not change. This proves the first case. In the second case, arc ik is present in one of the other $n - 1$ equations. Here we have two subcases based on whether pv_j can be written in terms of pv_k, the equation that contains arc ik. If not, it trivially does not affect security. Otherwise, that means that some pv_l (in the path between j and k) will have pv_k as one of its terms in equation 4.2. So, we note that if pv_k is reduced, the inner term of the product will increase as the total product, minimizing pv_l. Since we assumed that is possible to write pv_j in terms pv_l (pv_k), this ends the proof. ■

Before we continue, we add a further observation (Obs9) regarding fragilities. It is well accepted that software defects are a matter of bug density. Current industry numbers for delivered code estimate around five defects per kilo-line-of-code () with a cost of USD5 per line. Even extremely debugged software, as the space shuttle flying software has a density of about 0.004 bugs/kloc with a cost of USD850 per line of code [64]. Thus, it is a tempting idea that reducing the lines of code, while preserving the functionality, will improve security.

Policy 2: *Minimize the size of the Trusted Computing Base.* Explanation: The trusted computing base (TCB) is the set of components that provide secure services or protects system principals. This policy states that, in order to improve the overall system security, the components that constitute the TCB shall be reduced.

Theorem 2: Policy 2 does not always hold for integrity CSPs.

Proof: We use "Definition 2". It suffices to show that we can arbitrarily increase system security by increasing the size of the TCB. The size of the TCB in Definition 2 grows whenever the number of lines of code in V grows or when a new protective component

(vertex) is added. Let u be this new vertex and j the vertex to be protected. If arc uj exists, then, by Definition 2, it can either be associated in a "and" or a "or" relation. If it is associated in a "and" fashion, then, by equation 4.5, some $X \subset N_D^-$ will be added with in-arc uj, thus making the innermost product smaller and, as a consequence, total probability pp_j also smaller. This concludes the proof. ■

This negative result on Policy 2 asks for a more precise definition for TCB recommendation. We propose a more restrictive policy:

Policy 2 Reviewed: Given a system architecture, minimize the size of its individual components.

This policy is trivially supported by equation 4.5 and Obs9.

4.3.3 Other Considerations

Independence of Events

Although Observations 1 to 8, Properties 1 to 5 and Definitions 1 and 2 do not require independence of events, while writing the respective equations we limited ourselves to this case. We claim that this has limited effects on the previous proofs and the quantitative assessments obtained from these equations. In [57] it is shown that the main cause of correlate weaknesses is the gained knowledge by the adversary. In our model, we can easily compensate this fact by the addition of a further component representing the "lack of adversary knowledge" on a given weakness class. This is consistent with concept of obfuscation.

The Cost Function $c = f(p)$

The behavior of the function that maps probability of successful attacks to costs (property B4) is important for understanding the attacks themselves - know it lets us optimize the system architecture, reducing the likelihood of successful attacks while reducing costs. Nevertheless, the exact determination of the function $c = f(p)$ is probably an impractical task, as suggested by [70]. It happens, however, that we can establish a basic empirical behavior sufficient for our model: The domain of $f(p)$ is the closed interval [0,1] and image $\Re+$. We claim (without proof) that $p = f^{-1}(c)$ grows monotonically, a reasonable assumption based on the observation that the more resources placed in the attack, the greater the chance of success.

Cumulative Distribution Function for Definition 1

In Definition 1 we calculate the probability that an individual bit of S leak to the adversary. If we consider independent and equiprobable chances of leakage, the binomial

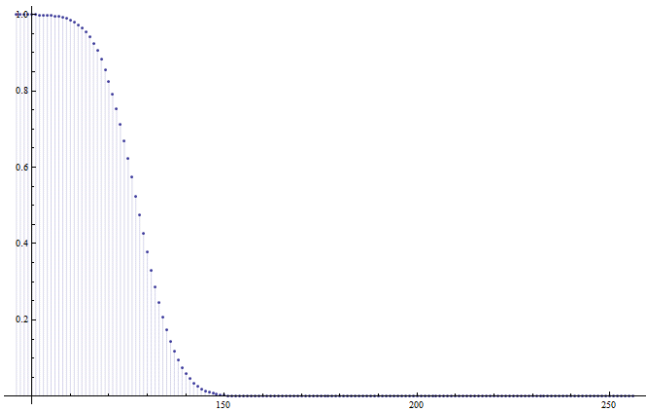


Figura 4.6: Complement of the CDF with parameters $p = 0.5$ and $n = 256$.

distribution tells us the expected number of leaked bits. In Figure 4.6 it is plotted the **complement** of the for the Binomial Distribution with $p = 0.5$ and 256 trials (bits). We learn that even for large probabilities of single bit leakage, the adversary will hardly learn sufficient bits to perform a brute force attack. This result is in line with the estimates of the actual behavior of p, which are surprisingly large, when physical attacks are considered.

4.3.4 DTProbLog Modeling

The convenience of using graphs to express Properties B1 to B5 and Observations 1-8 is impaired when the number of system components (and their interrelationships) grows and specializes, requiring multiple graphs to represent the same system or a considerable increase in the number of annotations. Moreover, the coding of the knowledge base of weaknesses, interaction channels, and probabilities is also hampered when using graphs.

For this reason, we coded the properties and observations of FORTUNA with the logical probabilistic inference language DTProbLog [68]. The DTProbLog is a recent extension of traditional Prolog that supports in its knowledge base probabilistic facts (e.g. 0.9: protects_directed (J, I)) and allows queries on probabilistic results. DTProbLog also has support for Decision Theoretic Problems, allowing the programmer to specify an optimization target (or decisions) (e.g. ? :: attacked(C) :- component(C) and utility functions (or costs and gains) (e.g. break_policy(C) => - 5 :- component(C)). Our application in DTProbLog allowed us to quickly and exactly perform the evaluations of our definitions. In the next section, we present the FORTUNA tool and a case use.

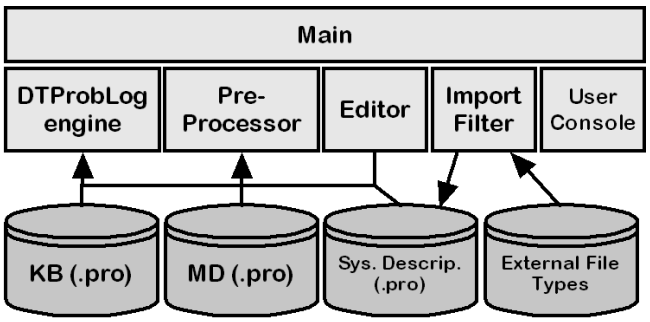


Figura 4.7: FORTUNA tool architecture

4.4 Implementation and Case Study

4.4.1 The FORTUNA Tool

The FORTUNA tool was implemented as a user application. Figure 4.7 shows the application architecture. The main components are MR (Model Rules) and KB (Knowledge Base), both DTProbLog program files. In MB are encoded variations of "Definition 1" and "Definition 2", along with supporting rules (as how to traverse our digraph and the rules to calculate the shortest path of Definition 2).

Component KB encodes the best currently knowledge about components, relations, costs and probabilities. KB was first generated using the defects/kloc metric: known software components had their lines counted. Physical protections had their numbers gathered from the literature. Hardware components had their number of lines of hardware description language counted or estimated (a complete SPARC processor, for instance has some 500.000 VHDL , as Gaisler's Leon 3 implementation).

Component "System Description (.pro)" contains the architecture of the current system under development. This file can either be directly edited by the user or automatically generated by the "import filter component" that processes external file types. Once the system description is complete, the "Pre-processor" component checks the file for common errors that prolog would not catch, as typos in relation names and components. When system description is ready, the user can load all DTProgLog programs and start asking queries through the user console, either queries from the MD or those built into DTProbLog:

```
?- dtproblog_solve(Strategy,ExpectedGain).
ExpectedGain = 17.12121,
Strategy = [attacked(coating),
attacked(die),attacked(priv_key)]
```

This represents the best attack strategy based on our models and the user system

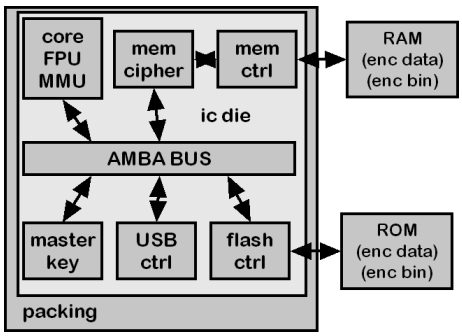


Figura 4.8: Former SCuP architecture, before FORTUNA

description. In particular, this attack, with positive gain, is viable. Thus, we must use this information to reinforce the components that take part in the attack strategy. Of course, by doing that the designer also needs to update the component utility functions to reflect the updated component cost.

4.4.2 Use Case

We employed FORTUNA in the conception and development phases of the Secure Cryptographic Micro-Processor (SCuP), which incorporates learnt lessons from our framework. When the development of SCuP started, it was conceived as a traditional secure processor as PC-TPM [32] or AEGIS [63], but, due to the threats pointed by FORTUNA, it evolved to an Asymmetric Multicore Processor (AMP) with novel features as the Hardware Firewall and the embedded Reference Monitor.

Figure 4.8 shows the simplified initial processor architecture (we omitted most software components). We employed FORTUNA in this architecture. The KB component was built using the bug density metric explained above and the system description was directly edited with the tool. Running FORTUNA we found the best attack strategy for the initial architecture:

```
?- problog_max(path(adversary,master_key),
        Prob, Strategy).
Strategy = [attacked(usb_ctrl),
        attacked(usb_stack),
        attacked(running_bin),
        attacked(master_key)]
Prob = 0.04809
```

This strategy, with calculated 4.8% chance of success, happens to be quite reasonable as USB stack problems were found and exploited in many operating systems. After multi-

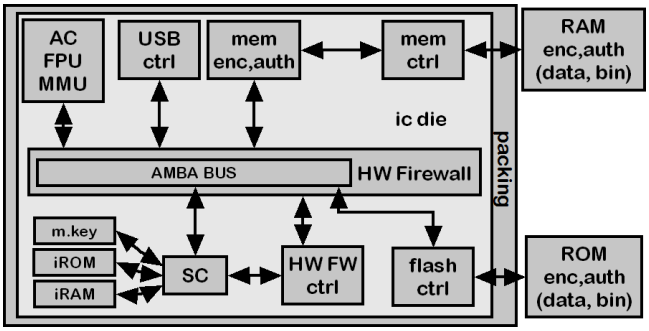


Figura 4.9: Former SCuP architecture after FORTUNA

ple FORTUNA runs and design changes, we ended up with the architecture of Figure 4.9. It has clear differences from the former architecture as many other protective components were added and the topology has changed.

The new architecture features two cores. The secure core () has reduced peripherals and software stack and is the sole responsible for handling the software's trusted base. The application core () runs business logic with SC monitoring (by using reference monitors [58]).

FORTUNA was employed in the middle of the design phase of SCuP and promoted important speedups in the weekly security design reviews. The work done in four hours could be carried out in less than 60 minutes. This clearly exemplifies the framework's usefulness.

4.5 Related Work

This framework is related to the following areas of secure architectures: hardware security, security metrics, risk assessment, patterns, and security policies, among others.

Hardware security has been addressed in many different forms. Security standards as Common Criteria [65] and FIPS140-2 [46] bring a large set of recommendations, requisites and policies a device may implement. In special, Common Criteria is concerned with the security evaluation of the target systems. Mirjalili and Lenstra [40] elaborate on broader security aspects of secure hardware by considering the entire life-cycle: design, manufacturing, user setup, repair and disposal.

Regarding security verification, our work is marginally related to formal system verification where logical components are formally described and formal proof techniques are employed to determine the validity of the descriptions. Although powerful, these techniques have two problems. First, they do not capture the stochastic nature of physical components; and second, they have complexity either NP-hard or undecidable [23, 27, 51],

which hinders their use.

As a consequence, totally informal or mixed techniques are employed[9] for logical validation. For instance, simulation-based techniques for logical security evaluation have been employed in the works by Payne [49] and Dwoskin [48].

Another related area is risk assessment. Schneier's seminal work on Attack Trees [59] discusses a broad-scope attack evaluation method based on tree representations. The idea was further improved by many other authors as in the PhD thesis by Schechter [57] and Manadhata [35] and Wang [71]. All these works deal with software or network security metrics and try to quantify security. However, the recent negative results (2009 and 2010) by Verendel [70] show that "Quantified Security is a Weak Hypothesis". This result was obtained after the analysis of nearly one hundred works in the area from 1981 to 2008. Accordingly, we pay more attention in our framework to qualitative aspects rather than exact values: this is totally compatible with the expected description levels available at early design time.

For the observations and properties of our model we used results from many different areas of security as secure co-processors [15, 63, 11, 66, 13, 60, 32], systems implementing reference monitors [58, 72, 50] and side-channel attacks [28].

4.6 Conclusion and Future Work

In this paper we presented the FORTUNA framework. With FORTUNA we bring both theoretical and practical considerations to the conception and early design stages of hardware-base secure systems. FORTUNA was successfully used to improve the architectural security of a cryptographic processor and to accelerate its design process. FORTUNA consists of two main elements. The first is a logical-probabilistic theoretic model with which we could prove the "least privilege policy" and obtain other qualitative results and insights. The second is an automated tool, based on the introduced model, that can assist in secure system design from the very initial development stages.

Although it is very practical and lightweight, there is certainly room for improvements in FORTUNA by, for example, improving its precision with the addition of the KB from other users. Also, improvements to the filter tool would allow for less user effort in qualifying components' relationships.

Future work includes the adaptation of the model equations to directly support conditional probability. Also, we are working on new policies based on our models and definitions.

Capítulo 5

SCuP - Secure Cryptographic Microprocessor

Publicação: Este artigo foi apresentado no "XI Simpósio Brasileiro em Segurança da Informação e de Sistemas Computacionais", na data de 6 a 11 de Novembro de 2011, em Brasília, Distrito Federal, evento da SBC, com publicação nos anais do evento.

Roberto Gallo[1,2], Henrique Kawakami[2], Ricardo Dahab[1]
(1) Campinas State University, Campinas, SP, Brazil
{gallo, rdahab}@ic.unicamp.br
(2) KRYPTUS Security Solutions Ltd, Campinas, SP, Brazil
{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.

5.1 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.

5.2 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 [15], 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 [63] 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 [11] 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) [66], 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 uti-

lizado 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 [13] 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 [60], 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 [32, 48]. 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 [32], 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 () não sofre das mesmas fragilidades que o sistema em verificação (), 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 [58, 32] 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 [72] e CoPilot [50], 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 re-

visitada 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* [58]). 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 [56]. 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 [47], assim como em [17, 20], 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 [40]. Neste sentido, padrões como o FIPS 140-2 [46] e o Common Criteria [65] 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 [23, 27, 51], dificultando o seu uso na prática. Desta forma, técnicas totalmente informais ou mistas são utilizadas na verificação das propriedades dos sistemas [9]. 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 [49] e Dwoskin [48].

5.3 Arquitetura do SCuP

A Figura 5.1 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 (e) 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.

5.3.1 Componentes do Sistema

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 5.1, $n = m = 1$. 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

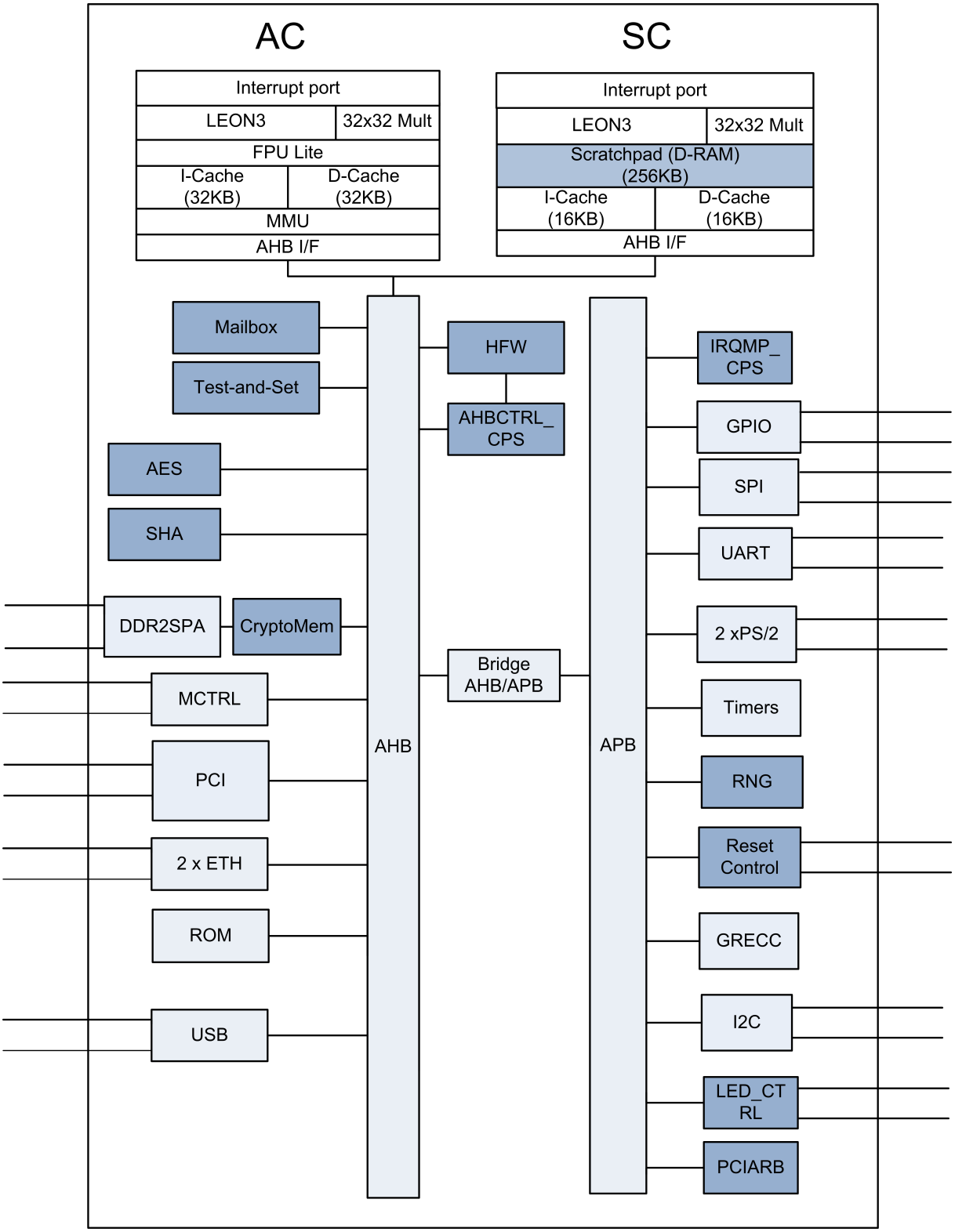


Figura 5.1: A arquitetura do SCuP mostrando os seus diversos módulos e periféricos.

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, e 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.

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.

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 - - com chaves de 256 bits em modo de operação não-padrão. O CryptoMem tem performance de processamento de 128 bits/ciclo.

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.

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.

5.3.2 Funcionalidades do SCuP

SBS - Sequência de Boot Seguro

A sequência de boot seguro () 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.

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.

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.

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 5.2: Consumo de elementos

5.4 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 5.2 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 () 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 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.

5.5 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.

Capítulo 6

Hardware Security: Towards a Holistic View or Extending FORTUNA

Publicação: Este artigo foi submetido ao *"Journal of Systems and Software"*, publicação da Elsevier, em agosto de 2012, e está aguardando revisão.

Roberto Gallo[1,2], Ricardo Dahab[1]
(1) University of Campinas, Campinas, SP, Brazil
{gallo, rdahab}@ic.unicamp.br
(2) KRYPTUS Security Solutions Ltd, Campinas, SP, Brazil
gallo@kryptus.com

Abstract

Security requires a holistic view. In this paper we investigate how a logic-probabilistic view of the problem of designing and developing secure, hardware-based, systems represents an important step towards achieving holistic security on this class of systems. Our contributions in this paper are: (a) extending and validating the Logic-Probabilistic Framework FORTUNA, and (b) showing the effectiveness of our methodology by finding a never reported SPARC V8 architectural flaw.

6.1 Introduction

6.1.1 Motivation

Computational systems are being plagued by security problem for decades. In the last fifteen years the once simple menaces evolved from sporadic manageable cases to a complex, ingenious, and highly risky scenario. This evolution was led by a natural, interactive process between "attackers" and "defenders"; the first always searching for the weakest link (the cheapest flaws to explore) and the seconds trying to make the offended links at least as secure as the others that form the security chain.

As security by itself is not the most common objective for a computational system, but rather a desired property (one expects that a given system abides by some security policy, if one exists), the evolution of the threats and the corresponding countermeasures in computational systems are subject to the frenetic race of the "performance versus price versus features" triple [40] which leaves little room for more drastic security improvements. In this scenario, typical security countermeasures are thought and implemented in a minimalist and localized fashion, creating a "patchwork" without a general and cohesive framework [3, 48].

This approach, however, is notoriously flawed as the conception of secure systems demands a global, holistic view. This is in part caused by the fact that the composition of individually secure components does not guarantee that the resulting set is itself secure[1]. To make things even worse, the complexity of information systems grows rapidly with respect to many dimensions: (a) the number of components, (b) the number of interactions with other systems, and (c) the class of system components. This complexity growth demands an expertise at the same time deep and multidisciplinary, hard to be attained and coordinated into a global view either by individuals or by a development team.

As a consequence, even when systems are thought from scratch [15, 63, 60, 32], the holistic view required for security is hard to attain. This difficulty may be partially accounted by the large number areas involved in computer security, which includes Discrete Mathematics, Statistics, Electronics, Physics, Computer Architectures, and Machine-Human Interfaces.

Security Views

It is relevant to mention the different security visions of the cryptology community and the security information community, as there are accentuated historical differences regarding

[1] It is usually not impossible to combine security policies in an additive manner. Also, greedy technics do not always lead to optimum results. Even more, the use of individually secure components is not even required to build a secure system, given a secure property

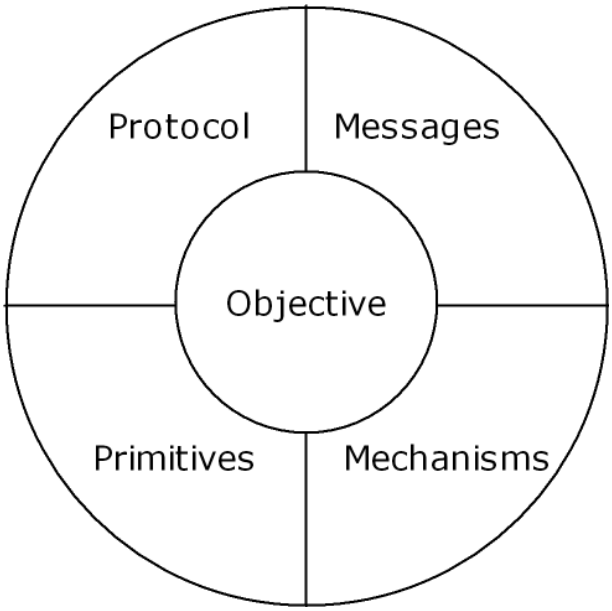


Figura 6.1: A cryptographic service completely covers the cryptographic objective.

security objectives and related techniques.

Cryptologists are typically interested in cryptographic services in order to achieve a certain **cryptographic objective**. Typical cryptographic objectives include confidentiality, data integrity, and authentication [39]. Cryptographic services are composed by protocols, messages, primitives and mechanisms that are designed, implemented and evaluated in a strict and formal fashion. Security proofs, given a security model (standard, random oracle, etc.), are sought (and many times found) for each individual element and their composition for the cryptographic service.

One says that a cryptographic service with a security proof (under a given security model) **has a certain security level** which expresses the expected workload (in terms of memory space or required steps) for an adversary in order to prevent the cryptographic objective to be achieved. The security level may be a function of many parameters.

For practical systems, typical security parameters are adjusted so that the security level is at least 10^{30}, requiring from the adversary an amount of resources or operations, even for the most powerful enemies.

In this sense, a cryptographic service completely covers the proposed security objective. Figure 6.1 illustrates this fact. Cryptographic services most of the time resembles a hermetic system. However, a potential problem with this approach is that if a single link is broken, the cryptographic objective is compromised as a whole.

Information security specialists, in turn, have a different view of the problem. Typically, their objective is to protect, at all costs, a given application (an user service or the

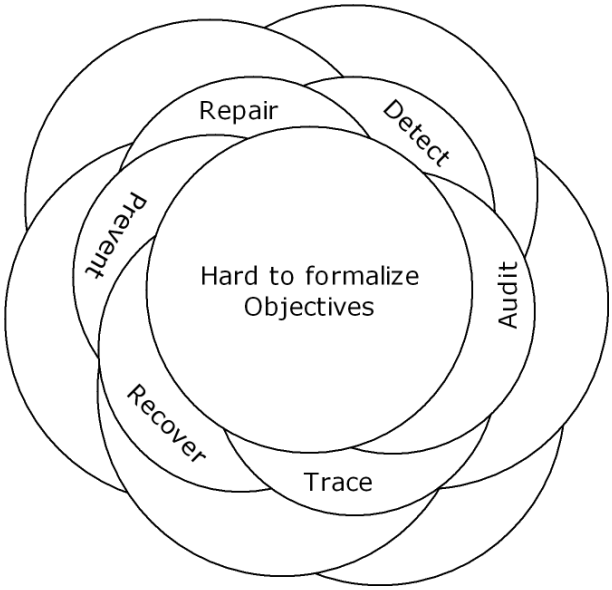


Figura 6.2: Security services are arranged in layers.

computational base itself) from internal and external adversaries and also from threats due to system defects, obeying an established security policy. Instead of having a specific and easy to describe objective, this community deals with a world that is hard to formalize, but where a failure or defect is a probabilistic certainty. Their objective is to guarantee the system dependability even knowing that their tools and systems under protection are incomplete, buggy and subject to failures[2]. There are no hermetic solutions.

In this scenario, where system comprise is almost a certainty, probability is a powerful and important analysis tool.

For these reasons, the way the security specialists target their objectives has a very different nature from the cryptographer's way; they achieve security by the development and composition of security services in layers, each one partially covering the security objectives. When existent, the security policy must describe those layers and corresponding security services. Figure 6.2 illustrates this approach.

Security services can be categorized in four classes:

- Detection services, which detect unexpected system behaviors that may indicate security problems (e.g. reference monitors[58]);

- Prevention services, which prevent or difficult certain adversary actions;

[2]Industry currently admits 5 bugs per thousand line of code with $5 cost per line. Even extremely debugged code, as the NASA Space Shuttle control software, where each line of code costs some $850.00, has a 0.0034 bug per kloc [64].

- Recover and repair services, which automatically recovers system functionality and states;
- Audit and tracking services, which provides the necessary evidences for system diagnosis.

Evidently, some of the above mentioned services can employ trusted cryptographic services as subcomponents. For instance, the detection service may use a cryptographic data integrity service to determine whether a given application binary (or data) was modified. **The truth, however, is that there is no general mapping between arbitrary cryptographic and security services**.

The different approaches of the security and cryptographers communities are symptomatic of a much larger problem; there is no single theory covering all security aspects of general, real, secure systems.

In spite of that, some authors have crossed the borders of these two worlds [43, 63, 32, 72, 36]. The concept of security fusion of state machines is a relevant example in the path of composing individually insecure components into secure systems. The side channel analysis, in its turn, yielded a better understanding of the relations between cryptographic and implementation aspects as practical realizations of crypto-systems are not secure as the predicted security level (under the given theoretical model), mainly because of poor system modeling[3].

And under this unconnected security visions, application developers (who provide value to the final customers) are the most affected as they assume the computing base and their services are secure.

The Peculiarities of Our Vision of the Problem

Trustworthy computing, in real, practical environments requires more than only cryptographic primitives, secure code execution architectures, security policies or model proofs. It requires also the acknowledgement and the inclusion of more practical factors.

In the first place, the admission that meant-to-be-secure systems are frequently broken due to technical advancements, design problems or implementation issues. **For all intents and purposes, the security breach of a non-trivial system is a certainty. It remains to know when, how, and with what impact**. We do recognize this characteristic as probabilistic: "what is the chance of this component be subverted and with which resources/time?." This vision is very different than the current employed by most proposals.

[3]One consequence is that the composition of cryptographic services with same objectives, which once was seen as "heterodox" are now not only accepted but some times, recommended. A good example is the use of two (or more) symmetrical encryption algorithms with full disk encryptors as TrueCrypt

Second, one must be aware of the economical aspect, as real systems have real costs: trade-off and security level balancing involve not only system components, but also, human-related costs as design, deployment and usage expenditures. Balancing usually means stopping to target hermetic solutions, even when they do exist.

6.1.2 Our Contribution

Our contributions in this paper are (a) extending and validating our Logical-Probabilistic framework FORTUNA [18], by explicitly taking into account time as a relevant security factor, (b) showing how FORTUNA helped us to find a never reported security issue with the SPARC V8 processor architecture.

These contributions are part of a larger objective; the development of a complete framework supporting the conception, design, development, and deployment of secure hardware-based systems backed by a unified, global security view. Naturally, such a vision ranges many different areas, from physical security to the human-machine interactions[4].

6.1.3 Paper Organization

In Section 6.2 we present our study object in further details: the Logic-Probabilistic Framework FORTUNA. Section 6.3 presents our contributions to the Framework. In Section 6.4 we describe the analysis of a industry standard processor, which led us to discover a novel fragility on the SPARC v8 processor architecture. Section 6.5 describes related work. In Section 6.6 we conclude and suggest future work possibilities.

6.2 FORTUNA

The FORTUNA framework is intended to help the early development stages of hardware-based secure systems by providing both a **theoretical model** and an **evaluation tool**. The goal is to build models for security assessment based on the architecture of the system under development, not vice versa. This is a good approach because it allows the system architect to focus efforts on the design of the solution, using his/her standard development tools instead of mentally-translating different systems representations (design vs. secure). This way, the developer can implement the changes suggested by FORTUNA quickly and easily. Moreover, when system modifications occur (upgrades, security patches), FORTUNA can be used to reevaluate security efficiently.

[4]Such a framework shall cover at least the following items: (i) a guide for threat modeling of hardware-base systems, (ii) a theoretical framework, (iii) a set of component and architectural patterns, (iv) and an evaluation tool

As presented in our original paper [18], the model is based on a set of **observations** and derived properties, rooted in very basic system characteristics:

- O1: a secure system can be composed of other systems (or components)[5].
- O2: the security of systems has a probabilistic nature. A special case is with software defects; they are a matter of bug density[6].
- O3: individually insecure (with respect to a given policy) components can be arranged in the form of a secure system[7].
- O4: secure components (with respect to a given policy) may be arranged into an insecure system.
- O5: ultimately, all components are physical. Logical components are abstractions represented in a particular physical component configuration (or state)[8][9].
- O6: There are no complete descriptions of non-trivial practical systems[10].
- O7: Every component has an associated cost for its deployment (e_{comp}).
- O8: Certain (typically local) components are associated with adversary rewards ($g_{comp,adv}$)[11].

Based on the above observations five properties were extracted and used to form the **bases** of the FORTUNA model:

- B1: **interaction channel**: every subsystem that can be composed with others has one interaction channel. This interaction channel may be a logical abstraction, providing a communication channel. The channel can be directed or not. Interaction channels can be classified hierarchically, regarding available description levels. The channel can be as generic as "mechanically connected" or specific as "TLS connection". Components can have multiple connections (even between a given pair).

[5]System and component examples are: processor, memory, metal case, cryptographic key, user, and algorithm implementation. This observation is related to the concept of security fusion.

[6]It is well accepted that software defects are a matter of bug density.

[7]Security policies can be composed. Related to security fusion

[8]Logical components are abstractions of physical properties where only a fraction of the available entropy is considered and the rest is ignored.

[9]The exploit of the ignored information present at the physical level allowed the discovery of side channel attacks [28].

[10]Both from a physical and practical perspective: at design-time, knowledge is partial and resources limited.

[11]e.g. user data, cryptographic keys, authentication data.

- B2: **entropic potential**: represents the information assets that generate benefits for the opponent. The benefits can be directly useful (e.g. a user's private key) or indirect (e.g. a key that wraps the private user key). Entropic potential is directly associated to Critical Security Parameters (CSPs) [46]. Measured in bits.

- B3: **entropic impedance (or resistance to leakage)**: quantifies the permeability of components and interaction channels to entropy. It measures how well the information present in a component is made available and is incorporated into others around it through interaction channels. It is given as the probability that a given entropy amount migrates in a given timeframe from a component A to a component B trough an interaction channel AB.

- B4: **implicit security**: components, with a certain security policy set are subject to different attacks. Each attack can affect one or more of the security policies. Each attack has a different cost and different success probability. Thus, one can associate to a given policy a pair (p, c). Intuitively, $c = f(p)$ or, in other words, the more resource one puts in the attack, the greater the odds of succeeding.

- B5: **security provided**: expresses the ability an (directional) interaction has of transporting the implicit security experienced by a component A to a component B. Together with the implicit security, it expresses the "protection relationship" (A protects B) or, in other words, how the security policy implemented by A affects B through an interaction channel.

These five properties describe many of the features of high-level architectures of secure systems, especially at the conception and design phases. They apply to both physical and logical systems and can be used to model systems at the earliest development phases.

In the original paper we presented two types of modeling based on the given observations (1 to 8) and bases (1 to 5). The first uses graphs (Section 6.2.1), mainly for easiness of interpretation, and the second uses ProbLog, a logical-probabilistic language (Section 6.2.3).

6.2.1 Graph Modeling

Figure 6.3 shows the simplified architecture of a standard cryptographic token, a simple secure system. The diagram brings information of both physical and logical nature. Rectangular items indicate physical components, trapezoids indicate data items, and double-walled rectangles represent procedures (firmware components). Arrows indicate explicit interaction channels. Channels can be either logical or physical.

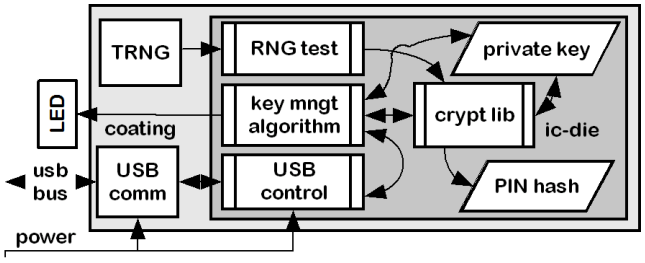


Figura 6.3: Graphical representation of a simple cryptographic token. The outer box represents the protective coating. The inner box, the IC die (microcontroller core) with logical components inside.

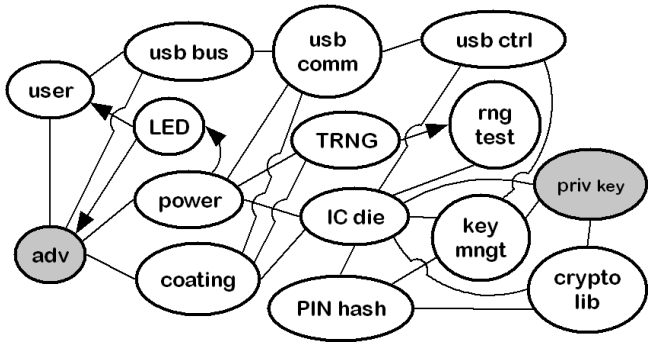


Figura 6.4: Graph representation of the same token architecture but obtained applying observations 1 to 8 and properties B1, B2 and B3.

Figure 6.4 shows the same architecture as figure 6.3 but with a graph representation, and adding user and adversary roles. All components are mapped to vertices. All interaction channels are mapped to edges (or arcs) between the components. The "contains" relation between the container and the contained components are mapped by connecting all contained items to the container (e.g. coating and TRNG, IC die and USB comm.).

Using properties B1 to B3, and marking "priv key" as the adversary's asset of interest, we can trace the ways in which this asset can leak to the opponent. An analogy that facilitates understanding is to resistive circuits (Kirchhoff's Laws): the private key represents the VCC and the opponent the GND. All edges are resistive. Vertices represent nodes. We are interested in an architecture that minimizes the entropic flow (current) "priv key" toward the opponent. Obviously, the analogy is limited since we do not have "the sum of current zero" but, instead, "the entropy of a channel is less than or equal to the sum of all entries".

Figure 6.4 facilitates qualitative security analysis. For example, it is easy to see that the logical architecture of the token does not maximize security: multiple edges incident to component "priv key"increase the likelihood of information leakage – there are multiple

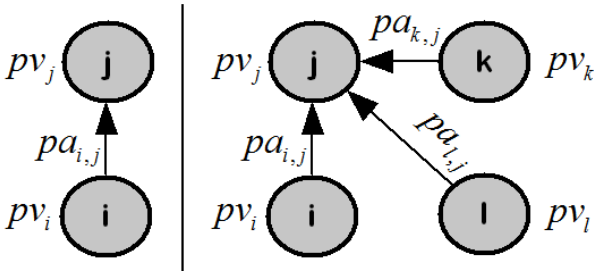


Figura 6.5: Definition 1: vertices and relations.

paths by which information can leak from "priv key" to the opponent. The same graphical model allows for immediate intervention: removing link between "key mngt alg"and "priv key"reduces risks. This modification conforms to the "least privileges policy". Formalizing all this, we have

Definition 1 (Model 1): Let $D = (V, A)$ be a digraph representing a related system and external agents that interact with it. V is the set of vertices representing the system constituents and external stakeholders (users, opponents). Let A be the set of arcs representing the interaction channel (B1). Let s be a bit of the secret S of interest to the opponent (B2) and which the system protects. By properties B1 and B3, we observe that s leaks from its container through the arcs with probability $pa_{i,j}$ (the probability that s transits from vertex i to j through arc ij), so that vertex j now has probability pv_j of knowing s. We are interested in minimizing pv for the vertex that represents the attacker.

In the following we consider all probabilistic events independent. Given Definition 1, pv_j in figure 6.5 is given by:

$$pv_j = pa_{i,j} \times pv_i, \tag{6.1}$$

where $pa_{i,j}$ is given. When j has multiple edges or incident arcs (Figure 6.5), it suffices that s leaks through only one arc; so, for each vertex j we can write:

$$pv_j = 1 - \prod_{i \in N_D^-(j)} (1 - pv_i \times pa_{i,j}), \tag{6.2}$$

where $N_D^-(j)$ is the set of all in-neighbors of j. For all n vertices of D we can write the same equality with a maximum of n variables in pv_i, which provides us with a system of degree n totally determined. For the vertex k that contains s, pv_k is trivially 1.

Figure 6.6 shows the same architecture of Figure 6.3. This digraph is a re-orientation of a subgraph of Figure 6.4, when applying properties B1, B4, and B5. This digraph represents the "protection" relationship and can be read alternatively as "the safety of vertex j depends on vertex i". We observe that the protection relations begin at the physical

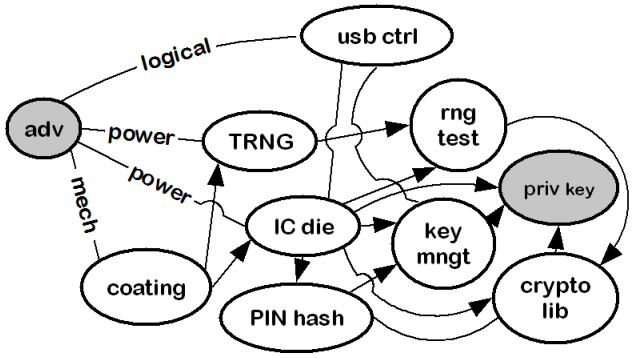


Figura 6.6: Protection (or security depends) relationship. Graph representation of the same token architecture but obtained applying observations 1 to 8 and properties B1, B4 and B5.

components and end at the logical ones, respecting the natural concept of "contained" - logical components are contained in one (or more) physical component(s). Interestingly, the protection relation has wider scope when compared with the entropy graph, as it can model different types of security objetives (e.g. integrity, availability), not only confidentiality. The representation of Figure 6.6 is related to Schneier's attack trees [59].

The graph of Figure 6.6 can be either split into a set of graphs, each for a security objective, or annotated so that it correctly captures security dependencies. In either case, it may require designer intervention to qualify the relations, especially regarding logical conditions (and, or)[12]. Obviously, for large systems, the required annotation amount may be impeditive and hinder the benefits of the graph analysis. For this reason, in Section 6.2.3 we present an alternative based on DTProbLog, a probabilistic derivation of the Prolog language. Now, we formalize:

Definition 2: Let $D = (V, A)$ be a connected digraph representing part of a system. V is the set of vertices representing components that establish relations of protection. By O8, for each vertex v of V there is a cost e_v (financial, work, performance) associated with the use of v. Let A be the set of arcs representing the protection relationships (or security dependence) via a given interaction channel. By B4, for each arc uv of A there is a cost c_{uv} associated with a given probability of success pp_{uv}. The arcs incident on v can be composed in and/or form. Let C be a subset of V representing the system's CSP. By O9, to each vertex v of C is associated a gain g_v. We are interested in making the best attack plan more expensive than the expected gain for the adversary.

In the following we consider all probabilistic events independent. Given Definition 2, the expected probability of success of an attack on vertex j of Figure 6.7 is given by $pp_j = p_{ij}$, while the expected cost of a successful attack on j via arc ij is given by:

[12]e. g. the security of component A depends on component B (and/or) C.

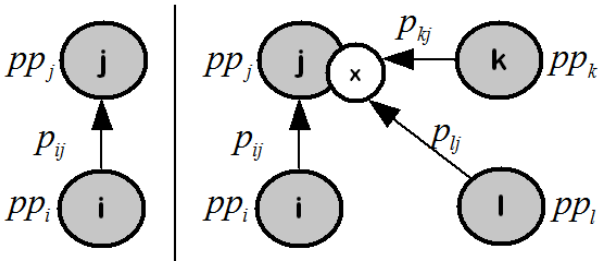


Figura 6.7: Definition 2: vertices and relations.

$$e_j = e_{ij} = \frac{f(p_{ij})}{p_{ij}} + e_i, \tag{6.3}$$

where p_{ij} and $f(p_{ij})$ are given and e_i is the cost of attacking vertex i. When multiple protection relations are established, the cost is given by the smallest attack value among all possibilities (incident arcs). The problem is therefore related to the "shortest-path problem", but with some changes: when an "and" relationship is established ($\times$ sign in the figure) all "shortest paths" must be accounted. We can write the relationship for each vertex in the graph as:

$$e_j = min\left(\sum_{x\in X} e_{xj}, \sum_{y\in Y} e_{yj}, ..., \sum_{z\in Z} e_{zj}\right), \tag{6.4}$$

where X, Y and Z represent sets of arcs grouped around the "and" notation and $N_D^-(j) = \bigcup X \bigcup Y \bigcup, ... \bigcup Z$ and $\oslash = X \bigcap Y, \vee X, Y$. Now, the probability that a given vertex is successfully attacked is given by:

$$pp_j = 1 - \prod_{X\in N_D^-}\left(1 - \prod_{i\in X} p_{ij}\right). \tag{6.5}$$

6.2.2 Graph Model Analysis and Results

The two models obtained from Observation 1 to 8 and Properties 1 to 5 allow not only for automatic security evaluation and design insights but it also permits theoretical exploration. In this section we prove two well-accepted security policies under our model.

Policy 1: *Grant to system principals the least privileges necessary to perform their jobs.* Explanation: this policy states that, in order to improve security, system principals, such as users and processes, must be granted access only to the resources needed for the task. That means restrictions on accessed data (user, system), cryptographic keys, processing and storage resources.

Theorem 1: Policy 1 either does not affect, or it improves the overall system security regarding confidentiality CSPs.

Proof: We use "Definition 1". The principals are represented by vertices that relate to (access) confidential resources (vertices that contain the s bit) via one or more arcs (either directly or chained). We want to minimize pv_j of equation 6.2 for the vertex that represents the adversary. Thus, it suffices to show that any arc uv removal never make pv_j larger. We have two cases based on whether the arc uv is present in the equation for vertex j or not. If it is, an arc removal implies that the product of equation 6.2 has one less iteration. Since each term $1 - pv_i \times pa_{i,j}$ is always equal to or smaller than 1, this removal either does not change the total value of the product or increases it. That means that the value for pv_j in equation 6.2 either diminishes or does not change. This proves the first case. In the second case, arc ik is present in one of the other $n - 1$ equations. Here we have two subcases based on whether pv_j can be written in terms of pv_k, the equation that contains arc ik. If not, it trivially does not affect security. Otherwise, that means that some pv_l (in the path between j and k) will have pv_k as one of its terms in equation 6.2. So, we note that if pv_k is reduced, the inner term of the product will increase as the total product, minimizing pv_l. Since we assumed that is possible to write pv_j in terms pv_l (pv_k), this ends the proof. ■

Given observation O2, it is a tempting idea that reducing the lines of code, while preserving the functionality, will improve security. The policy 2 captures this idea:

Policy 2: *Minimize the size of the Trusted Computing Base.* Explanation: The trusted computing base (TCB) is the set of components that provide secure services or protects system principals. This policy states that, in order to improve the overall system security, the components that constitute the TCB shall be reduced.

Theorem 2: Policy 2 does not always hold for integrity CSPs.

Proof: We use "Definition 2". It suffices to show that we can arbitrarily increase system security by increasing the size of the TCB. The size of the TCB in Definition 2 grows whenever the number of lines of code in V grows or when a new protective component (vertex) is added. Let u be this new vertex and j the vertex to be protected. If arc uj exists, then, by Definition 2, it can either be associated in a "and" or a "or" relation. If it is associated in a "and" fashion, then, by equation 6.5, some $X \subset N_D^-$ will be added with in-arc uj, thus making the innermost product smaller and, as a consequence, total probability pp_j also smaller. This concludes the proof. ■

This negative result on Policy 2 asks for a more precise definition for TCB recommendation. We propose a more restrictive policy:

Policy 2 Reviewed: Given a system architecture, minimize the size of its individual components.

This policy is trivially supported by equation 6.5 and O2.

6.2.3 DTProbLog Modeling

The convenience of using graphs to express Properties B1 to B5 and Observations 1-8 is impaired when the number of system components (and their interrelationships) grows and specializes, requiring multiple graphs to represent the same system or a considerable increase in the number of annotations. Moreover, the coding of the knowledge base of weaknesses, interaction channels, and probabilities is also hampered when using graphs.

For this reason, we coded the properties and observations of FORTUNA with the logical probabilistic inference language DTProbLog [68]. The DTProbLog is a recent extension of traditional Prolog that supports in its knowledge base probabilistic facts (e.g. 0.9: protects_directed (J, I)) and allows queries on probabilistic results. DTProbLog also has support for Decision Theoretic Problems, allowing the programmer to specify an optimization target (or decisions) (e.g. ? :: attacked(C) :- component(C) and utility functions (or costs and gains) (e.g. break_policy(C) => - 5 :- component(C)). Our application in DTProbLog allowed us to quickly and exactly perform the evaluations of our definitions. In our original paper, we presented the FORTUNA tool and a case use that we do not reproduce here.

6.3 A Temporal Extension for FORTUNA

Time is a fundamental variable for the security analysis under FORTUNA's core properties and is implicitly linked to each of them. We comment on the reasons and on the impacts of time on each of the five properties presented in section 6.2.

6.3.1 B1 and B5: Interaction Channel and Security Provided

Whenever the system architecture changes over time, the interaction channel changes. This occurs when a user plugs an USB device on his computer or simply when a new process or thread is started on a time-shared processor. Depending on the description level of the target system and the accuracy of probabilities and costs (B3, B4), these changes may be more or less relevant to the security analysis. Of special impact are the cases when the architectural change implies in related modifications of the entropic potential (B2) or the protection relationships (B5).

Figure 6.8 presents a simple target system: a microcontroller with an internal program memory (uC/ROM), volatile memory (RAM), display interface, a plugged pen drive and two users: a legitimate and an adversary both with the same interfaces. At initial time $t = 0$, the pen drive stores a key (with probability $P(key) = 1$). The arrows show directed interaction channels (B1).

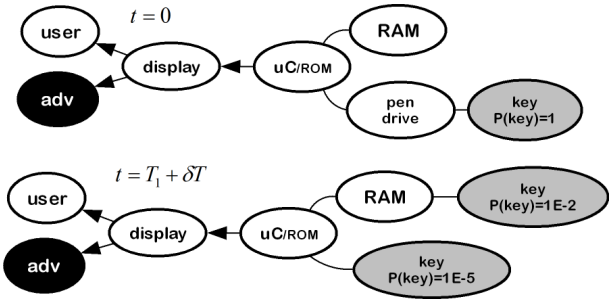


Figura 6.8: Simple architecture at different times.

After some time, at $t = T_1$, the key as has a probability of **being seen** by other components, say vertex j, given by equation 6.2. The farther the component is from the source, the smaller the probability pv_j.

If at $t = T_1 + \delta T$ the pen drive containing the key were removed, then the system architecture and interaction channels would be modified. As a result, equation 6.2 (and eventually equation 6.5 for protection) must be re-evaluated.

Were we assured that no information was **learnt** by any other system component, then pv_j for all vertices would be trivially zero. However, in most cases, certainty of not learning is an exception and the figure shows a possible outcome that embraces "learning". Learning has a probability different from merely "seeing" the passing data and figure 6.8 capture this concept by adding new "probabilistic components"[13].

6.3.2 B2 and B3: Entropic Potential and Entropic Impedance

The entropic potential measures (in bits) the information (probably) known by the given component. It is typically a function of the time: memory contents vanishes without refresh, semiconductors "may permanently learn" bits stored for long periods and system architectures often move data around the system (sometimes leaving multiple data copies, other times erasing it). The exact time-dependent behavior of the entropic potential depends, of course, on each component's specificity.

However, some component characteristics have clear impact on the time behavior:

- **Memory capacity**: components with explicit memory capability, as RAM, caches, register files or process variables are more prone to learn data than stateless components as buses, IC coatings, and program text segments.

[13]Given the model from Definition 1, we have at least two valid ways to represent learning. The first is by adding new components (vertexes) that have probability of knowing $pv_j \leq 1$. Another possibility is by adding edges with $pa_{i,j} \leq 1$. We prefer the first because it makes graphically easier to understand that the key is spreading.

- **Exposition time**: the longer the data is presented to the component, high the probability that this value will be available at a subsequent moment[14].

- **Total data influx**: when the influx has probability of been tainted by the secret value (low entropic impedance), the larger the data amount the higher the probability that the target component will learn the sensitive information. However, when non-tainted data is available, large influxes in fact diminishes the learning probability as they "wash-out" the target component.

- **Addressable versus stream component**: stream components as FIFOs are much more prone to "forget" the data once learnt, than addressable components as RAM memories. This is true because exciting all control signals in a memory mapped device is more difficult, so some data may be left behind.

The first three characteristics when considered together resemble an electric capacitor that may charge (with tainted information) and discharge (with non-tainted information). When helping designer to concept and design systems at early stages, it is important to identify and avoid components and architectures that charge rapidly and discharge slowly[15] and have large number of control signals as addresses.

Other Considerations

Mapping every component aspect that affect the time behavior is a complex task and ultimately depends on modeling the entire component itself, which is not our aim with this work. However, mapping more common components as PROM memories, for instance, has clear value and could be done by means of multiple time representations with architectural changes like those in figure 6.8.

While modeling systems by graphs (as in Definitions 1 and 2) may be very instructive, more than just a few time-frames may be too expensive in terms of workload. However, this is feasible with DTProbLog (the basis of our tool) as the language allows for conditional statements and automated evaluation.

If we had we continuous or semi-continuous time-dependent descriptions of the target components, then it would be possible to describe the system as a set of differential equations and then use numerical techniques to determine the overall system behavior. We anticipate, however, that we would need to change the probabilistic analysis domain to a data flow domain.

[14]Intuitively, the information has more time to spread inside the component internals.
[15]Now it is clear that the probabilities $P(key)$ in figure 6.8 are time dependent.

6.3.3 B4: Implicit Security

The relation between probability and cost is of course dependent of the time available to the adversary to perform the attack. The shorter the available time, the more expensive the attack is likely to be.

6.4 Results

6.4.1 SPARC Architectural Vulnerability

The SPARC processor architecture was once one of the main players in the market in terms of performance and sales. Nowadays, this architecture remains relevant thanks to the architecture specifications public release and to the industrial use by important vendors as SUN/Oracle, Fujitsu, and Gaisler Research, among others. As these processors are commonly employed in critical systems as database servers, mainframes, super-computers, space launchers and even satellites, security is paramount.

The SPARC architecture has two principal versions the V8 introduced in the beginning of the 1990's and the V9 later in the same decade. Employed with the Solaris Operating System (marketed by former SUN Microsystems) this platform has a positively recognized security history.

In [19] we employed the SPARC V8 architecture for the development of a secure cryptographic microprocessor (SCuP) with help of the original FORTUNA. As both the framework and the SCuP continued to evolve, the introduction of further tool reports led to identify what later was shown to be a major security issue. This issue allows for data leakage among same-process procedures and even among different processes and it is contained in the SPARC V8 and V9 architecture for now 30 years. To the best of our knowledge this is the first time this issue is reported.

Entropic Tool for Capacitance - ETC

The FORTUNA Entropic Tool for Capacitance - ETC - is an analysis tool that processes semi-automatically[16] the HDL synthesis report and HDL files to identify components that demand security designer attention.

For that end, we use the heuristics given by equations 6.6 and 6.7 to estimate the time dependent "capacitive" effect of the target component, as discussed in section 6.3. The first equation is intended to assign scores to components that have addressable storage

[16]The tool requires human intervention to distinghish between data and control (and address) lines and to define the component hierarchical level to be considered, among others.

whereas the second equation is intended to stream components. The scores represents the expected stored bits at given a time t.

$$S_{addr}(t) = \begin{cases} c_s \times c_{\#} \times \left(1 - \left(\frac{c_{\#}-1}{c_{\#}}\right)^{t \times w_r}\right), & \text{if input data is tainted} \\ c_s \times c_0 \times \left(\frac{c_{\#}-1}{c_{\#}}\right)^{t \times w_r}. & \text{if input data is clean} \end{cases} \tag{6.6}$$

$$S_{stream}(t) = \begin{cases} c_s \times t \times w_r, & \text{if input data is tainted} \\ c_0 - c_s \times t \times w_r. & \text{if input data is clean} \end{cases} \tag{6.7}$$

with $0 \leq S_{stream} \leq m$

In the above equations, S is the total capacity score (in bits) and t represents time. We also consider components with m-bit storage capacity written each time in chunks of size c_s, thus components comporting $c_{\#}$ chunks of data. Since data for most components are written at different rates, w_r is necessary and represents the data chuck write rate. Both c_s and w_r are component and system dependent parameters and are themselves functions of numerous factors such as (a) the relation between m and the size of the width of input ports, (b) how fast new data is admitted in the input signal, and (c) how frequently data is written into the input port. If time t is measured in clock cycles then, for stream components, w_r has magnitude of the unit, which is intuitive and expected for components such as FIFOs. If the frequency in which data is written to the input ports of the particular component is known, either by simulation, by inspection of similar systems, or any other means, the values for w_r and c_s may be adjusted in the FORTUNA tool knowledge database accordingly. For memory mapped devices with random access w_r may have values close to the unit, however c_s may vary greatly but in a quite predictable fashion[17].

Theorem 3: Equation 6.6 expresses the expected updated bits on a randomly written memory addressable device with fixed-chunk data length.

Proof: The proof is straightforward if we note that only the last value written on a given memory position matters as the previous content is overwritten. Given that, the probability of a given position **not** being written when a single write occurs is given by $\frac{c_{\#}-1}{c_{\#}}$. After $t \times w_r$ writes, the probability is reduced to $\left(\frac{c_{\#}-1}{c_{\#}}\right)^{t \times w_r}$. When the input is not tainted, the initial c_0 chunks vanish similarly, proving the-non tainted case. When the input data is tainted, we need to consider only updated chunks. As the complement of

[17]The variation is not difficult to estimate: dynamic allocated variables typically use fixed-size escalating bucket buffers, as in most POSIX malloc implementations whereas statically allocated variables are typically small multiples of the machine's word width. For caches, c_s is the cache line length and for registers the machine's word width.

Tabela 6.1: ETC output for the Leon II SPARC V8 employed in SCuP with $w_r = 1/4$, $c_s = 32$

rank	component	as stream (charge-discharge)	as addr (charge-discharge)	user
1	cache (cmem0)	651266/571266	651266/575982	–
2	usb device (grusbdc0)	133319/53319	**133319/73157**	no
3	usb host (usbhc0)	36410/0	**36410/4041**	no
4	reg file (rf0)	16443/0	**16443/126**	yes
5	fpu (fpu0)	**18872/0**	18872/271	yes
6	pci dma (pcidma0)	5051/0	**5051/0**	no
7	ddr crtl (ddrc0)	1357/0	1357/0	–
8	ethernet ctrl (eth0)	**729/0**	729/0	no

not being written is the probability of being written at least once, the expected updated chunks are given by $c_{\#} \times \left(1 - \left(\frac{c_{\#}-1}{c_{\#}}\right)^{t \times w_r}\right)$. This concludes the proof. ■

With the standard Leon processor (SPARC V8 architecture) we are interested in studying inter-process security; in our scenario a target process handles sensitive information for a given time (the typical intra-schedule grant time, some 100 milliseconds yielding 10×10^6 instructions), "charging" components when the processor control is passed to the kernel for process scheduling (typical time 10 - 100 microseconds), discharging components when a malicious thread is scheduled for execution.

Table 6.1 shows score information generated by ETC from synthesis output report generated by the Altera Quartus II v11.1 for a Stratix III FPGA for the top 8 scored processors components (from over 20 components). The table present the component score rank, the component type and instantiation name, the score when classified as stream component, the score when classified as memory addresable component, and whether the component is directly accessible by user-space programs. Numbers in bold reflect how the component was classified, no bold means no classification.

The table clearly shows that four components have large probability of holding sensitive data between target and exploit processes. The large amount of information stored in the data cache was already expected, but is a problem of less magnitude, as process insulation is already commonplace in this component and user process cannot access arbitrary cache pages. Both USB components (device, host controller) can hold large amounts of data as they must have internal buffers in order to achieve the necessary performance, so they can be used to leak data between processes. However, in most operating systems the address space of these components is kernel protected and no direct access is possible. Designers should take extra caution when writing device drivers for the USB stack since

user space program can exploit software defects.

The register files, differently from other components, can be directly accessed by user programs. A brief analysis shows that the maximum score for the register file presents a surprisingly large number, 16443 bits when we expected roughly 32 (regs) x 32 (bits) = 1024 bits. Also, intuitively we would expect that after a few cycles most information would have been discharged but the abnormal final score of 126 bits is almost the size of an AES key bits. This anomaly prompted for the deeper analysis of the processor register file.

Register Window

The existence of a register window as register cache mechanism was only "discovered" after the manual inspection of the processor architecture [61, 62] and is responsible for the anomalies found by ETC. In this architecture, each register file has 32 immediately addressable registers with a configurable circular register window which can have from 2 to 32 levels (NWINDOWS architecture parameter). These 32 registers are divided into 4 sets of 8 elements: a **global set**, common to every window, a **local set** reserved to each window, an **in set**, shared with the **out set** of the previous window. The out set is shared with the in set of the next register window. Figure 6.9 shows the register window. The register window moves when instructions SAVE, RESTORE and RETT are issued or when a trap occurs.

Thus, the number of available platform registers is not 32 as one could learn from the inspection of some processor's instructions, but it is given by the 8 + 2 x 2 x 16 x NWINDOWS, where the default value for NWINDOWS is 7. This results in 230 registers, which explains the anomaly found by ETC.

Register Dive Exploit

Our exploit is accomplished by carefully deepening into the register windows without destroying their contents in a kind of dive. By doing that, we were able to access data left by other procedures. In a system with an O.S. this means access to kernel data or even other users' processes information. Figure 6.10 illustrates our attack.

To confirm this fragility we have implemented an example exploit in assembly which was latter executed by the Gaisler Research Leon 3 SPARC V8 Commercial Core. With this implementation we were able to recover the contents of past procedures executions confirming the foreseen fragility whose signs were shown by the FORTUNA ETC tool.[18].

[18]Special thanks to Watson Yuuma Sato for the proof of concept implementation

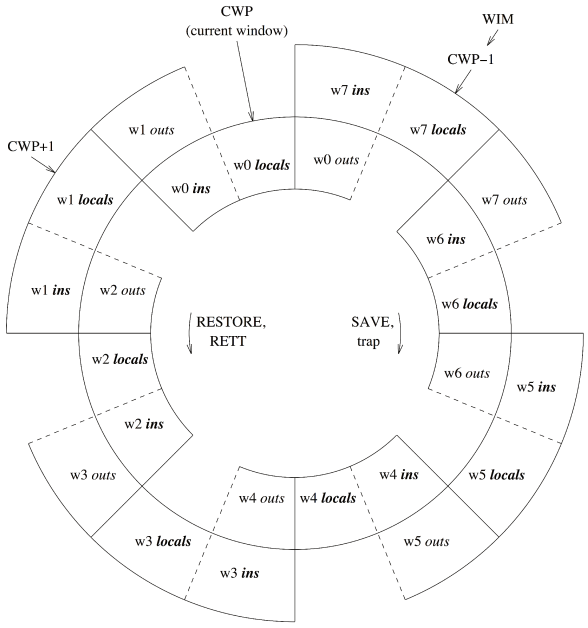


Figura 6.9: SPARC V8, V9 register window (from [61]). The SPARC specification imposes no cleaning mechanisms to the non-used registers, leaving to the compiler this task. We use this characteristic in our exploit.

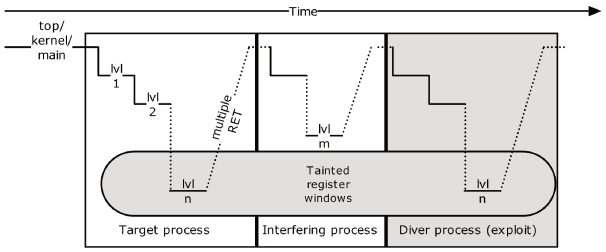


Figura 6.10: Register dive exploit. Information from the target process (or procedure) taints the register windows. This data however may be available to an exploiter process even when some other process is run. A diver process can thus descent to a register window and recover previously stored data.

6.5 Related Work

Our work is related to a significant number of computing areas: i) secure hardware architectures, ii) secure co-processors, iii) prevention, detection, and recovery of security violations, and iv) security metrics.

Secure Co-Processors and Safe-Code Execution

Most relevant related work include the seminal design of the IBM 4758 [15], which first presented many security mechanisms, both logical and physical later employed in many devices. The IBM4758 is a PCI board, multi-chip, co-processor that works as a hardware security module running (signed) user programs under an 80486 CPU.

Although highly secure, the IBM 4758 is unsuitable for many use cases; for this the AEGIS architecture [63] represents an important evolution as it is a single component processor able to perform safe code execution employing a trust of chain originating on the boot component. Moreover, the processor has a modified scheme for process insulation. The AEGIS off chip RAM, in turn, is completely encrypted and authenticated.

The Cerium processor [11] is another relevant proposal, although less comprehensive, which features an interesting benefit: certified application output. This represents a shift on the stakeholder position; integrity may be of interest to a third party as in DRM (digital right management) applications.

DRM applications are subject to a particularly interesting threat model; on one hand the content (music, movies, applications) production may cost millions of dollars but in the other hand the individual gain of the common adversary is just a few dollars; gains and losses are highly decoupled.

This threat model was prevalent at the conception of the Trusted Platform Module (TPM) from the Trusted Computing Group [66], which is a platform (hardware and software) for trusted computing both in personal computers and mobile devices. The TPM itself is typically presented as a peripheral soldered to the computer motherboard. This device has functionalities for digital signature, system state measurement and state attestation, so that individual applications or the entire boot chain can be verified.

This proposal has several virtues i) very low cost (few dollars), ii) do not require legacy code rewriting, iii) provides effective protection against binary image (.text) manipulation. The down sides are: i) it is a slave device so it can be completely ignored, moreover, it cannot start a system integrity inspection autonomously, ii) low computational power and slow data bus preventing massive usage, iii) easily subverted by tampering only with as small part of the unprotected BIOS, the CRTM, iv) provides little support for confidentiality.

In order to improve the security profile without considerable expenditures Costan et al [13] proposed and implemented (using Java Cards) the Trusted Execution Module (TEM) which is able to execute safe code inside the module using the Secure Execution Closure Packs (SECpakcs). SECpacks allow application of arbitrary complexity to be rewritten into smaller packs and them executed inside the TEM. Of course, it implies in additional I/O operations and performance degradation.

The secure execution packs may date to the IBM4758 proposal, but only with the IBM

Cell [60], employed in the Sony Playstation 3, they were deployed more consistently. Cell is an asymmetrical multi-core processor specially designed for the entertainment industry where processing power and content protection are both paramount. Of special interest are the Synergistic Processing Elements (SPE) which carry both the high performance computation but can also be set to run in secure mode, called *secure processing vault* mode (with both data and code signed and encrypted). In this mode, the core runs insulated from the other cores so that no other core is able to inspect or modify data or code under execution. The core is slave of no other core. The benefits are clear: augmented security against direct attack to contents and improved resistance against escalation (through attacks coming from other cores).

These safe execution packs may be seen as a special type of thread insulation or safe thread execution where the number of threads running simultaneously is limited to the number of cores: normal threads does not coexist with secure threads.

The execution of secure (or insulated) threads simultaneously with regular ones was first implemented in the AEGIS proposal by means of privileged execution modes. This was precisely what the more recent SP architecture presents [32, 48]. The SP proposal is both more general and minimalist and can be implemented in virtually any general-purpose processor with just a few interventions. The SP architecture requires changes to the target processor instruction set and addition of memory encryption components. By employing the proposed Trusted Software Module (TSM) along with the hardware modifications, the SP can provide confidentiality and attestation services, among others.

Architectures for Monitored Execution

Although not pointed by Lee [32], we can conjecture that with modifications on the TSM, the SP architecture could be employed for software inspection/introspection. This usage, however, has leverage only if the verifying system does not suffer from the same fragilities of the systems under verification.

In order to minimize this possibility much is said about the minimization of trusted computing base where the software stack (BIOS/boot loader, operating system, secure applications) are reduced to a few thousands of lines of code.

However, to the best of our knowledge, no author has expressed concern about the fact that modern processors (and related digital circuitry) are described in hundreds of thousands or even millions of lines of hardware description languages. This is even worse if we consider that current synthesizers are unaware of security of the generated systems. For that reasons it is a temerity to expect that the hardware base has no hidden security problems due to implementation issues.

Works as CuPIDs [72] and CoPilot [50] are results of a different research avenue: they employ pairs of systems, one security (policy) monitor and another monitored. CoPilot is

aimed at monitoring and recovery from rootkit attacks. It is implemented by means of a PCI-master device (monitor) connected to a host (monitored system) so that the monitor is able to inspect the target's entire memory space.

The monitor does not share resources with the monitored system; this way, in case of a rootkit installation on the main system, the PCI board is able to verify that modifications occurred in the kernel memory space and as consequence warns the stakeholders and even repair the tampered system. **Because monitor and monitored have completely different architectures the possibility that they share weaknesses is minimized.** The CoPilot limitations are related to the hardware cost and performance degradation due to bandwidth and processing power of the monitor.

In CuPIDS the idea of independent systems for monitoring is reviewed leading different hardware and software approaches. By using a motherboard with two processors, the authors split the system into a monitor portion and a monitored portion. In contrast to CoPilot, the authors propose for each service (on the monitored portion) a co-service responsible for monitoring the EM-enforceable [58] security policy. By using different software stacks and processors monitor and monitored portions are less likely to be tampered with by a single thread. However, since they are implemented in general-purpose processors, attacks such as binary replacement are still possible.

Our more recent SCuP [19] proposal explicitly bridges some of the gaps between safe code execution, secure-processors and monitored execution. By using asymmetric multi-core processor architecture, based on SPARC, SCuP presents high-speed monitoring capabilities, secure boot sequence, encrypted off-chip memory and tight peripheral access control.

System Integrity and Verification, Standards

When integrity of the offered services is mandatory and mission critical, different techniques and approaches have been used to achieve high assurance levels[19]. In these situations it is admitted that systems are (never) 100% trustworthy and can be tampered with. As this, effective integrity verification mechanisms must be present so that stakeholders can assess the security levels[20].

Other relevant factor for system integrity relates to the trustworthiness of the system life-cycle as reported by Mirjalili and Lenstra [40], which start at the system conception and goes until the system is disposed. In this sense, industry and governmental standards

[19]One of the most demanding applications is electronic voting. Such systems must achieve simultaneously seven objectives: one vote per voter, vote registered as marked, vote counted as registered, vote secrecy, vote verifiability and coercion resistance [56]

[20]The interaction between systems and stakeholder where explored in works as VoteBox Nano [47] and Gallo et al [17, 20].

such as FIPS 140-2 [46] and Common Criteria [65] play major roles. FIPS 140-2 presents a set of requisites and recommendations that a cryptographic module (for generation, storage and use of cryptographic keys) should observe. Although no particular system architecture is mandated by the standard, it presents a comprehensive list of security goals such as logical and physical protections and auto-tests. The Common Criteria is a meta-standard that defines templates with items that Protection Profiles must cover. A protection profile describes the expected characteristics of the target system (and many related processes), which later can be "certified" under different levels of assurance.

Regarding system (security) verification, our proposal is marginally related to works that use formal tools. In these works, systems and expected security characteristics are described in some formal language so that proofs are sought trying to validate or refute the proposed security features. Although powerful, these techniques are NP-hard (or even non decidable [23, 27, 51]), hindering their usefulness. As a result, totally informal or hybrid techniques are been employed [9], some times using simulation, sometimes using virtual machines Payne [49] and Dwoskin [48].

A completely different approach is the use of attack trees [59], a seminal concept introduced by Schneier. Instead of trying to model the entire system, this concept is used to model the paths and actions an adversary need to perform in order to achieve his goals. The attack trees later evolved to attack graphs in a number of works, mainly related to software security.

6.6 Conclusion and Future Work

In this paper we presented an extended version of FORTUNA, a probabilistic framework for conception and early design stages of hardware-based secure systems. To the best of our knowledge, FORTUNA is the only framework to assist the security assessment of hardware-based system implementations.

For this, we employed a probabilistic approach based on five defined properties and nine observations. With these properties, two graph-based models were built. These models allowed us to prove or defy well-accepted security policies regarding system security: the least privileges policy is one of such examples.

Also, we extended our original work by explicitly considering time: to better model target systems, it is important to support system descriptions (connections, components) that change over the time. For this, we proposed the concept of entropic capacitance, a fundamental connector between evaluated time-windows and, using this concept, we developed the FORTUNA ETC tool. Precisely this feature led us to discover a never reported security architectural flaw in the SPARC V8 and V9 processor architecture that we could explore. The uncovering of this security issue states the usefulness of our

framework.

However useful, that discovery required manned actions as verifying whether the observed anomaly could indeed be exploited and required a great deal of creativity.

Our current and future efforts are concentrated in the next step of the development of secure systems: determining whether a software-hardware complete specification has flaws based on a language approach. Instead of trying to uncover never-seen flaws we will leverage on extensive security flaw publicly available and use pattern-matching techniques to identify if the target architecture suffers from any known vulnerability. In some sense our methodology resembles an anti-malware but target at hardware description.

Capítulo 7

Conclusão

Resumo dos Resultados

Apresentamos nos capítulos anteriores trabalhos que estabelecem o nosso *framework* para o desenvolvimento de sistemas seguros baseados em hardware. As contribuições estão organizadas de forma cronológica e refletem o amadurecimento de nossa metodologia. Um sumário dos elementos do *framework* está disponível na Seção 1.5.2.

No Capítulo 2, introduzimos diversos conceitos centrais no nosso trabalho, dentre eles o do *CID*, sua ligação com o ciclo de vida de sistemas seguros, sua integridade física e lógica, e a interação com usuários por meio de um esquema de amplificação de confiança. Os conceitos ali apresentados foram implementados em um HSM real para experimentação. Boa parte dos resultados foram incorporados na versão de produção do equipamento ASI-HSM, que hoje contém, gerencia e protege as chaves da Autoridade Certificadora Raiz da Infraestrutura de Chaves Públicas Brasileira (AC-Raiz da ICP-Brasil), AC-Receita Federal, AC-SERPRO e AC-NIB.BR/DNSSEC, dentre outras ACs, com histórico de segurança perfeito. O ASI-HSM foi o primeiroHSM homologado para uso na ICP-Brasil e é o único com mais alto nível de segurança previsto: NSF2-NSH3.

No Capítulo 3, avançamos nos resultados do Capítulo 2, utilizando e estendendo os resultados conceituais anteriormente introduzidos, além de propor novos componentes para o nosso *framework*: a análise da segurança dos canais de verificação humana, e uma nova arquitetura de segurança com controle de acesso a periféricos controlados por hardware com privilégios baseados em certificados digitais. As novidades apresentadas foram implementadas em um protótipo de bancada de urna eletrônica e mais tarde incorporadas em sua maioria à Nova Urna Eletrônica Brasileira (modelo 2010), com 400 mil equipamentos fabricados (em contratos de mais de R$400 milhões entre TSE e Diebold-PROCOMP). Deste total, 200 mil urnas já foram utilizadas na eleição brasileira nacional de 2010. Nenhum caso de fraude foi apurado nessas eleições. Estes fatos dão amplo suporte às nossas

propostas.

No Capítulo 4, estabelecemos, com o FORTUNA, os fundamentos teóricos do nosso framework. FORTUNA é baseado em um conjunto de observações e propriedades, utilizado no estabelecimento de três modelos lógico-probabilísticos: i) baseado em grafos para relações de proteção; ii) baseado em grafos para modelagem entrópica; iii) baseado em linguagem lógico-probabilística de uso geral. Sob estes modelos, políticas de segurança como "mínimos privilégios" foram demonstradas, enquanto outras, como "minimização da base computacional confiada", foram desafiadas. Com o modelo baseado em linguagem construímos uma ferramenta de análise de segurança de sistemas utilizada como suporte no desenvolvimento do SCuP, um processador seguro. No nosso entendimento, FORTUNA é o primeiro, e até este momento único, framework para concepção e desenvolvimento inicial de hardware seguro.

No Capítulo 5 apresentamos o SCuP, um processador criptográfico com execução segura de código, i.e., cifrada e 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 seguirá para difusão semicondutora nos próximos meses[1]. O SCuP reúne em um único sistema-alvo grande parte dos elementos de nosso framework.

No capítulo 6, revalidamos e estendemos a contribuição do FORTUNA ao considerar, agora de maneira explícita, a dimensão temporal na análise de sistemas seguros. Em especial, introduzimos métricas para avaliação de elementos dos sistemas-alvo mais propensos a causar problemas de segurança por meio da ferramenta ETC. O uso da ferramenta ETC serviu de guia para encontrarmos uma fragilidade arquitetural inédita nas especificações dos processadores SPARC V8 e V9. Em uma implementação de prova de conceito do *exploit* pudemos recuperar informações (centenas de bits) entre processos sem uso de código privilegiado. Esta descoberta valida o FORTUNA como ferramenta de auxílio na concepção e projeto de sistemas seguros.

Perspectivas para Trabalhos Futuros

Além dos trabalhos futuros listados em cada um dos artigos apresentados[2], os nossos próximos passos na área de sistemas seguros baseados em hardware estarão concentrados na criação de uma ferramenta capaz de identificar automaticamente vulnerabilidades em

[1]Este projeto tem o apoio do programa de subvenção econômico da FINEP.

[2]Alguns daqueles trabalhos indicados já foram realizados nos artigos subsequentes

sistemas já totalmente desenvolvidos, por meio da busca de padrões de vulnerabilidades previamente colhidas, catalogados a partir de outros sistemas.

Esse trabalho é objeto de estágio de pós-doutoramento deste autor. Quando finalizado, este trabalho poderá ampliar o escopo de nossos resultados.

Os principais desafios relacionados com esta linha de pesquisa incluem a definição do que vem a ser uma vulnerabilidade no contexto de interação entre hardware e software, além da definição de classificadores de vulnerabilidades. Ambas definições deverão ser posteriormente utilizadas para a análise automática de descrições de hardware à procura de padrões que representem possíveis problemas de segurança.

Considerações Finais

Quando iniciei a pesquisa em segurança da informação em hardware, há quase 13 anos, ainda como estudante de graduação, a área apresentava um grande vazio de proposições em todos os sentidos: mapeamento de vulnerabilidades, conhecimentos sobre ataques, propostas de soluções, arquiteturas, modelos e teorias que explicassem mais detalhadamente as razões por trás dos problemas de segurança.

Somente na década passada, nos anos 2000, começaram a aparecer os primeiros trabalhos mais específicos sobre segurança em hardware, cobrindo uma gama mais ampla de problemas de segurança, com os primeiros coprocessadores seguros. Estes primeiros resultados, de autoria de grupos da IBM, do MIT e da Princeton University, pareciam apontar para um futuro com soluções bastante herméticas de segurança ao considerar múltiplos mecanismos que protegiam contra a grande variedade de ataques conhecidos.

A realidade no entanto, se mostrou mais dura do que se pensava inicialmente: a ideia de que é possível mapear fragilidades e depois agir sobre elas individualmente foi invalidada à medida que os tipos e mecanismos de ataques sobre sistemas baseados em hardware foram se multiplicando. Ficou claro que é fundamental que o "desconhecido" seja levado em conta quando pensamos em segurança.

Ainda que o desconhecido possa ter origem no descobrimento de novas técnicas de ataques (como o advento dos ataques de canais colaterais), a maior fonte de fragilidades parece advir dos defeitos ("bugs") de software, que em muitos casos implica em comportamentos inseguros dos sistemas alvo. O problema com os defeitos, no entanto, é que eliminá-los é uma tarefa próxima do impossível e conviver com eles talvez seja a única opção no paradigma atual de computação.

Conviver com tais defeitos pode significar muitas coisas, desde projetar sistemas de software que não apresentem pontos únicos de falhas, até sistemas que possam recuperar-se de situações inseguras com o auxílio de uma infraestrutura de hardware. A primeira opção não parece promissora pois a tríade desempenho-potência-custo representa um grande im-

peditivo. A segunda opção, por outro lado, é aquela que acaba de ser adotada pela gigante Intel-McAfee que finalmente trouxe para o mercado em um só conjunto algumas das tecnologias que foram isoladamente propostas pela comunidade científica nos últimos 10 anos sob o codinome Intel IPT. Estas tecnologias incluem modos seguros de processamento, caminhos de dados seguros, aceleradores criptográficos embarcados nos processadores e mecanismos de inspeção de software com suporte em hardware.

A adoção destas tecnologias tem o potencial de mudar drasticamente o cenário de segurança computacional nos próximos anos, de maneira nunca vista. Nos sistemas menos críticos, os problemas diários de vírus e "root-kits", roubos de internet banking e aplicações maliciosas roubando dados de outras aplicações poderão virtualmente desaparecer, tirando muito da pressão atual sobre soluções de segurança das plataformas computacionais isoladas (computadores quando pensados de forma isolada). Estes fatos irão deslocar os ataques e assim os problemas de segurança para sistemas formados de grandes composições de plataformas heterogêneas, para a nuvem e para o fator humano.

Nos sistemas críticos, onde há adversários motivados, os problemas deverão continuar amplamente em aberto, esperando por uma mudança de paradigma ainda por vir. Para estes sistemas o que foi proposto pela indústria é claramente insuficiente.

Glossário

AC do inglês *Application Core*, núcleo de execução de aplicações gerais proposto por nós para o processador SCuP.

AC de Autoridade Certificadora, entidade emissora de certificados digitais no contexto de infraestruturas de chaves públicas.

ACSAC2010 Vigéssima sexta conferência anual sobre aplicações em segurança em Austin, Texas, com publicação pela ACM.

AEGIS é uma arquitetura de processador seguro.

AES do inglês *Advanced Encryption Standard*, cifrador de blocos especificado no NIST FIPS PUB 197.

AHB do inglês *AMBA High-performance Bus*, tipo de barramento de alta performance previsto no padrão AMBA.

ALUT do inglês *Adaptive Look-Up Table*, unidade de elemento configurável de determinadas FPGAs da fabricante Altera.

APB do inglês *Advanced Peripheral Bus*, tipo de barramento para periféricos previsto no padrão AMBA.

API do inglês *Application Programming Interface*.

ARM do inglês *Advanced RISC Machines* é tanto uma família de arquiteturas de processadores RISC como o nome da empresa que as desenvolver.

AVR do inglês *Atmel's Alf (Egil Bogen) and Vegard (Wollan)'s Risc processor*, arquitetura de microcontrolador com conjunto reduzido de instruções.

BIOS do inglês *Basic Input Output System*.

BSI do alemão *Bundesamt für Sicherheit in der Informationstechnik*, órgão do governo alemão.

CA do inglês *Certification Authority*, autoridade certificadora no contexto de infraestrutura de chaves públicas.

CDF do inglês *Cumulative Distribution Function*, função distribuição acumulada, no contexto da Estatística.

CID do inglês *Cryptographic Identity*, identidade criptográfica de um dispositivo, apresentado por nós no artigo do EuroPKI'09.

CPA do inglês *Correlation Power Analysis*, técnica de SCA baseada na análise de correlação estatística de potência consumida pelo dispositivo alvo.

CPU do inglês *Central Processing Unit*.

CRTM do inglês *Core Root of Trust for Measurement*, porção de código que forma a raiz de confiança em um sistema com o módulo TPM.

CSP do inglês *Critical Security Parameter*, parâmetro crítico de segurança, como definido no padrão NIST FIPS PUB 140-2.

DMK do inglês *Device Master Key* é uma (ou mais) chave criptográfica utilizada em diversos sistemas como raiz de confiança e sobre a qual a cadeia de confiança do dispositivo e de seus serviços é construída.

DPA do inglês *Differential Power Analysis*, técnica de SCA baseada na análise diferencial de potência consumida pelo dispositivo alvo.

DRE do inglês *Digital Recording Electronic*, denota máquina de votação digital (urna eletrônica).

DRM do inglês *Digital Rights Management*, sistema digital de gestão de direitos (autorais), tecnologia usualmente empregada na proteção de mídias.

DSK do inglês *DRE Signature Key*, chave (assimétrica) da máquina de votação, utilizada para certificar a ESK, ambas previstas pelo VVSG.

EA do inglês *Electoral Authority*, autoridade eleitoral, no Brasil representada pela Justiça Eleitoral (TSE e TREs).

ECC do inglês *Elliptic-curve Cryptography*, criptografia de curvas elípticas.

ESK do inglês *Election Signature Key*, chave (assimétrica) de votação, utilizada em um único pleito e prevista pelo VVSG.

ETC do inglês *entropic tool for capacitance*, é uma ferramenta que faz parte do *framework* FORTUNA.

EuroPKI2009 Sexto workshop europeu sobre infraestrutura de chaves públicas e aplicações em Pisa, Itália e com publicação pela Springer na série LNCS.

EVS do inglês *Electronic Voting System*, sistema eletrônico de votação.

FINEP de Financiadora de Estudos e Projetos, órgão a União.

FIPS do inglês *Federal Information Processing Standard*, série de padrões do órgão americano NIST.

FIPS 140-2 é um padrão publicado pelo NIST para módulos de segurança criptográficos.

FPGA Field-programmable gate array.

FPU do inglês *Floating-Point Unit*, unidade aritmética de ponto flutuante.

FRAM do inglês *Ferroelectric RAM*, RAM estática ferroelétrica.

GRUB do inglês *GNU GRand Unified Bootloader*.

HID do inglês *Human Interface Device*, dispositivo de interface homem-máquina.

HMAC do inglês *Hash-based Message Authentication Code*, código de autenticação com chaves baseado em funções de resumo criptográficas, definido no padrão NIST-FIPS-PUB 198.

HSM ou MSC (módulo de segurança em hardware), do inglês *Hardware Security Module*.

HSSRTC do inglês *High Stability Secure Real Time Clock*, relógio real seguro de alta estabilidade, por nós proposto.

HWF do inglês *hardware firewall*, é um mecanismo de proteção de faixas de endereços de memória presente no processador seguro SCuP.

HWM do inglês *Hardware Modification*.

IBM International Business Machines Corporation.

IBM4758 é um HSM da IBM.

INPI de Instituto Nacional da Propriedade Industrial.

IPT do inglês *Intel Privacy Technologies*, conjunto de tecnologias de segurança embarcado nos processadores intel a partir da linha *Ivy Bridge*.

ISO International Organization for Standardization.

IVS do inglês *Independent Verification Sheet*, folha de senhas contendo TOTVs para uso individual.

kloc do inglês *kilo-line-of-code*, milhar de linhas de código.

KXT do inglês *Key eXtracTion from hardware*.

LCD do inglês *Liquid Crystal Display*, écran de cristal líquido.

LED do inglês *Light-Emitting Diode*, diodo emissor de luz.

LOC do inglês *Lines-of-Code*, unidade de linhas de código.

MCU do inglês *Microcontroller Unit*, microcontrolador.

MIP de Mecanismo de Inspeção/Introspecção Profunda, mecanismo proposto por nós no artigo do SCuP.

MIPS do inglês *Microprocessor without Interlocked Pipeline Stages*, é uma arquitetura de microprocessadores.

MMU do inglês *Memory Management Unit*, unidade de gerenciamento de memória.

MSM do inglês *Master Security Module*, módulo de segurança mestre por nós proposto para a nova urna eletrônica brasileira.

NIST do inglês *National Institute of Standards and Technology, USA*.

NRE do inglês *Non-Recurring Engineering*, custos não recorrentes de engenharia.

NSF2-NSH3 é o nível máximo de certificação de módulos criptográficos previsto no Manual de Condutas Técnicas número 7 da ICP-Brasil.

NSS2011 Quinta conferência em segurança de redes e sistemas, Milão, Itália, com publicação pela IEEE.

OATH do inglês *Initiative For Open Authentication*.

OTP do inglês *One-Time Password*, senha de uso único.

PCB do inglês *Printed Circuit Board*, placa de circuito impresso.

PCI do inglês *Peripheral Component Interconnect*, é um padrão de conexão de dispositivos periféricos.

PES de Pacotes de Execução Segura, vide SECPacks.

PKI do inglês *Public Key Infrastructure*, infraestrutura de chaves públicas.

PP do inglês *Protection Profile*, perfil de proteção do padrão Common Criteria.

PU do inglês *Processing Unit*, unidade de processamento no contexto de dispositivos seguros de hardware.

PUF do inglês *Physical Unclonable Function*, denotam construções físicas (de hardware) que apresentam características de comportamento únicas entre dispositivos mesmo que fabricados pelo mesmo processo.

RAM do inglês *Random Access Memory*, memória de acesso aleatório.

S-HI-HWM do inglês *Human Interface Secure up to Hardware Modification*.

S-HI-KX do inglês *Human Interface Secure up to Key eXtraction from hardware*.

S-HI-SWM do inglês *Human Interface Secure up to Software Modification*.

S-SVS do inglês *Simple Shared Verification Scheme*, esquema simples de amplificação de confiança proposto por nós.

SBS de Sequência de Boot Seguro, proposta por nós e empregada no SCuP.

SBSeg Simpósio Brasileiro de Segurança da Informação e de Sistemas Computacionais, promovido pela SBC.

SC do inglês *Secure Core*, núcleo de execução segura proposto por nós para o processador SCuP.

SCA do inglês *Side-Channel Analysis*, análise de canais colaterais utilizado como ferramenta de ataque de sistemas seguros.

SCuP do inglês *Secure Cryptographic Microprocessor*, processador criptográfico seguro proposto por nós no artigo de mesmo nome.

SDE do inglês *Secure Device Epoch*, conceito por nós proposto e que indica o instante de tempo do estabelecimento da identidade criptográfica (CID) de um dispositivo.

SECPacks do inglês *Secure Execution Closure Pack*, são pacotes de execução apresentados na solução TEM. São equivalentes ao PES apresentados em nosso artigo SCuP.

SEV de Sistema em Verificação, o mesmo que VDE.

SHID do inglês *Secure HID*, dispositivo de interface segura homem-máquina.

SI do inglês *Software Independence*, independência de software, conceito introduzido por Rivest para denotar sistemas de votação cuja a segurança não dependa da inexistência de defeitos de software.

SM do inglês *Signature Module*, módulo de assinatura preconizado pelo VVSG.

SPA do inglês *Simple Power Analysis*, técnica de SCA baseada na análise simples de potência consumida pelo dispositivo alvo.

SPARC do inglês *Scalable Processor Architecture* é um arquitetura de processadores RISC.

SPE do inglês *Synergistic Processing Element*, elemento de processamento do IBM Cell.

SPI do inglês *Serial Peripheral Interface bus*, padrão de barramento serial.

SRAM do inglês *Static RAM*, RAM estática.

SSCD do inglês *Secure Signature Creation Devices*, PP para dispositivo seguro com capacidade de criação de assinaturas (digitais).

SU do inglês *Sensing Unit*, unidade de sensoriamento no contexto de dispositivos seguros de hardware.

SV de Sistema Verificador, o mesmo que VRE.

SVS do inglês *Shared Verification Scheme*, esquema de verificação compartilhado com capacidade de amplificação de confiança, por nós proposto.

SWM do inglês *Software Modification*.

TCB do inglês *Trusted Computing Base*, base confiada de computação.

TE-SVS do inglês *Traitor Evidencing Shared Verification Scheme*, esquema de amplicaficação de confiança com capacidade de detecção de traidores, proposto por nós.

TEM do inglês *Trusted Execution Module*.

TLS do inglês *Transport Layer Security*, são protocolos criptográficos que conferem segurança de comunicação na Internet para serviços como email, navegação por páginas e outros tipos de transferência de dados.

TOE do inglês *Target Of Evaluation*, objeto alvo de avaliação pelo Common Criteria.

TOTP do inglês *Time-Based One-Time Password*, senha de uso único baseada em tempo.

TOTV do inglês *Time-Based One Time Verification code*, esquema de verificação de uso único baseado em tempo por nós proposto.

TPM do inglês *Trusted Platform Module*, é uma plataforma de hardware definida pelo *Trusted Computing Group* com capacidade criptográfica principalmente voltada para o uso em aplicações em DRM.

TRNG do inglês *True Random Number Generator* é o nome dado aos geradores de números aleatórios baseados em fontes físicas de entropia.

TrustZone é uma tecnologia para execução segura de código proposta pela ARM e composta de uma API e arquiteturas de referência.

TSE de Tribunal Superior Eleitoral.

TSM do inglês *Trusted Software Module*, módulo de software confiado apresentado no *framework* SP.

USB do inglês *Universal Serial Bus*, padrão de barramento serial de comunicações.

VDE do inglês *Verified Entity*, entidade em verificação participante de um esquema de amplificação de confiança.

VRE do inglês *Verifier Entity*, entidade verificadora participante de um esquema de amplificação de confiança.

VVSG do inglês *Voluntary Voting System Guidelines*, guia americano de recomendações para sistemas de votação.

Referências Bibliográficas

[1] ML Akkar, R Bevan, Paul Dischamp, and Didier Moyart. Power Analysis, What Is Now Possible... *Advances in Cryptology*, pages 489–502, 2000.

[2] R. Anderson, M. Bond, J. Clulow, and S. Skorobogatov. Cryptographic processors—a survey. *Proceedings of the IEEE*, 94(2):357–369, 2006.

[3] R. J. Anderson. 'trusted computing' frequently asked questions v1.1, 2003.

[4] Ross Anderson and Markus Kuhn. Tamper resistance—a cautionary note. USENIX, November 1996.

[5] Ross J. Anderson. *Security Engineering: A Guide to Building Dependable Distributed Systems*. Wiley, 2 edition, April 2008.

[6] Diego Aranha, Marcelo Karam, Andre Miranda, and Felipe Scarel. Vulnerabilidades no software da urna eletronica brasileira, v1.0.1, 2012.

[7] ARM Limited. TrustZone API Specification Version 3.0, February 2009.

[8] Brazilian Superior Electoral Court (TSE). Election statistics, April 2010.

[9] 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.

[10] D. Chaum. Secret-ballot receipts: True voter-verifiable elections. *IEEE Security & Privacy*, 2(1):38–47, 2004.

[11] 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.

[12] M.R. Clarkson, S. Chong, and A.C. Myers. Civitas: A secure voting system. 2007.

[13] 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.

[14] Conrado Gouvea Diego Aranha. Relic is an efficient library for cryptography. http://code.google.com/p/relic-toolkit/, April 2010.

[15] 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.

[16] A.J. Feldman and Josh Benaloh. On subliminal channels in encrypt-on-cast voting systems. In *Proceedings of the 2009 conference on Electronic voting technology/workshop on trustworthy elections*, pages 12–12. USENIX Association, 2009.

[17] 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.

[18] Roberto Gallo, Henrique Kawakami, and Ricardo Dahab. FORTUNA - A Probabilistic Framework for Early Design Stages of Hardware-Based Secure Systems. In *Proceedings of the 5th International Conference on Network and System Security (NSS 2011)*. IEEE, 2011.

[19] Roberto Gallo, Henrique Kawakami, and Ricardo Dahab. SCuP - Secure Cryptographic Microprocessor. In *SBSEG 2011*, nov 2011.

[20] 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.

[21] Rosario Gennaro, Anna Lysyanskaya, Tal Malkin, Silvio Micali, and Tal Rabin. Algorithmic Tamper-Proof (ATP) Security: Theoretical Foundations for Security against Hardware Tampering, 2004.

[22] A. Huang. Keeping Secrets in Hardware: The Microsoft XBox TM Case Study. *Cryptographic Hardware and Embedded Systems-CHES 2002*, pages 355–430, 2002.

[23] 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.

[24] ICP-EDU. Emissão de certificados pela AC raiz. http://www.icp.edu.br/svn/docs/template-emissao-cert-ac-credenciada.pdf, August 2009.

[25] International Organization for Standardization (ISO). *ISO/IEC 27002 Information technology — Security techniques — Code of practice for information security management*. ISO/IEC, July 2005.

[26] International Organization for Standardization (ISO). *ISO/IEC 11889:2009 Information technology – Trusted Platform Module*. ISO/IEC, 2009.

[27] 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.

[28] Marc Joye. *Basics of Side-Channel Analysis*, pages 365–380. Cryptographic Engineering. Springer, 1 edition, 2009.

[29] S. Kent. Evaluating certification authority security. volume 4, pages 319–327. IEEE.

[30] KRYPTUS, RNP, LabSEC/UFSC. *ASI-HSM Datasheet*, 2012.

[31] Leslie Lamport, Robert Shostak, and Marshall Pease. The byzantine generals problem. *ACM Transactions on Programming Languages and Systems*, 4:382–401, 1982.

[32] 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.

[33] Ralph Levien. *Computing with Social Trust*, chapter Attack-Resistant Trust Metrics, pages 121–132. Springer, London, England, November 2008.

[34] Ralph Levien and Alexander Aiken. Attack-resistant trust metrics for public key certification. In *7th USENIX Security Symposium*, pages 229–242, 1998.

[35] Pratyusa K. Manadhata and Jeannette M. Wing. An attack surface metric. *IEEE Transactions on Software Engineering*, 37:371–386, 2011.

[36] Stefan Mangard, Elisabeth Oswald, and Francois-Xavier Standaert. One for All - All for One: Unifying Standard DPA Attacks. Cryptology ePrint Archive, Report 2009/449, 2009.

[37] Jean Everson Martina, Túlio Cicero Salvaro de Souza, and Ricardo Felipe Custódio. Openhsm: An open key life cycle protocol for public key infrastructure's hardware security modules. In *EuroPKI*, pages 220–235, 2007.

[38] Maxim Integrated Products Inc. Usip-pro component datasheet, April 2010.

[39] Alfred J. Menezes, Scott A. Vanstone, and Paul C. Van Oorschot. *Handbook of Applied Cryptography*. CRC Press, Inc., Boca Raton, FL, USA, 1st edition, 1996.

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

[41] D. M'Raihi, M. Bellare, F. Hoornaert, D. Naccache, and O. Ranen. RFC 4226: HOTP: An HMAC-based one-time password algorithm, December 2005.

[42] D. M'Raihi, S. Machani, M. Pei, and J. Rydell. RFC draft: TOTP: Time-based one-time password algorithm, January 2009.

[43] Suku Nair, Subil Abraham, and Omar Al Ibrahim. Security fusion: a new security architecture for resource-constrained environments. In *Proceedings of the 6th USENIX conference on Hot topics in security*, HotSec'11, pages 2–2, Berkeley, CA, USA, 2011. USENIX Association.

[44] C. A. Neff. Practical high certainty intent verification for encrypted votes, October 2004.

[45] C.A. Neff. A verifiable secret shuffle and its application to e-voting. In *Proceedings of the 8th ACM conference on Computer and Communications Security*, page 125. ACM, 2001.

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

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

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

[49] 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.

[50] 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.

[51] 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.

[52] Eric Rescorla. Understanding the security properties of ballot-based verification techniques. In *Electronic Voting Technology Workshop / Workshop on Trustworthy Elections*, August 2009.

[53] Ronald L Rivest and John P Wack. On the notion of "software independence" in voting systems. *System*, 2006.

[54] Pankaj Rohatgi. *Improved Techiniques for Side-Channel Analysis*, pages 381–406. Cryptographic Engineering. Springer, 1 edition, 2009.

[55] Daniel Robert Sandler. *VoteBox: A tamper-evident, verifiable voting machine*. PhD thesis, Rice University, April 2009.

[56] Naveen K. Sastry. *Verifying security properties in electronic voting machines*. PhD thesis, Berkeley, CA, USA, 2007. Adviser-Wagner, David.

[57] Stuart Edward Schechter. *Computer Security Strength & Risk: A Quantitative Approach*. PhD thesis, Cambridge, Massachusetts, 2004.

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

[59] Bruce Schneier. Attack trees, 1999.

[60] 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.

[61] SPARC International Inc. *The SPARC Architecture Manual Version 8*, 1991.

[62] SPARC International Inc. *The SPARC Architecture Manual Version 9*, 1994.

[63] 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.

[64] Balachander Swaminathan. Agile overview, 2007.

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

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

[67] USA Election Assistance Commission. Recommendations to the EAC voluntary voting system, guidelines recommendations, 2007.

[68] G. Van den Broeck, I. Thon, M. van Otterlo, and L. De Raedt. DTProbLog: A decision-theoretic probabilistic prolog. In *Proceedings of the AAAI Conference on Artificial Intelligence (AAAI 2010)*, July 2010.

[69] Marten van Dijk, Jonathan Rhodes, Luis F. G. Sarmenta, and Srinivas Devadas. Offline untrusted storage with immediate detection of forking and replay attacks. In *STC '07: Proceedings of the 2007 ACM workshop on Scalable trusted computing*, pages 41–48, New York, NY, USA, 2007. ACM.

[70] Vilhelm Verendel. Quantified security is a weak hypothesis: a critical survey of results and assumptions. In *Proceedings of the 2009 workshop on New security paradigms workshop*, NSPW '09, pages 37–50, New York, NY, USA, 2009. ACM.

[71] Lingyu Wang, Tania Islam, Tao Long, Anoop Singhal, and Sushil Jajodia. An attack graph-based probabilistic security metric. In Vijay Atluri, editor, *Data and Applications Security XXII*, volume 5094 of *Lecture Notes in Computer Science*, pages 283–296. Springer Berlin / Heidelberg, 2008.

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

[73] Yu Zheng, Dake He, Hongxia Wang, and Xiaohu Tang. Secure drm scheme for future mobile networks based on trusted mobile platform. In *Proceedings of the IEEE International Conference on Wireless Communications, Networking and Mobile Computing 2005 (WCNM05)*. IEEE Press, Elsevier EI, 2005.