Segurança Cibernética em Smart Metering

Charles B. do Prado, Luiz Fernando R. C. Carmo, Davidson R. Boccardo, Raphael C. S. Machado, Lucila M. Bento, Cássia F. Novello, Alvaro E. Rincon, Luciana V. Matieli, Renato A. de Oliveira

***Resumo* – Medidores inteligentes são dispositivos dotados de software embarcado que realizam, em muitos casos, processamento de dados com um alto grau de complexidade. Estes equipamentos tem impulsionado o surgimento de um grande número de novas aplicações, mas por outro lado introduzem novos desafios com respeito a validação destes equipamentos. Este artigo descreve um projeto de P&D desenvolvido pelo Inmetro (Instituto Nacional de Metrologia, Qualidade e Tecnologia) em parceria com a Eletrobrás Distribuição Rondônia que envolve linhas de pesquisas voltadas para a validação dos medidores inteligentes.**

***Palavras-chave* – Medidores inteligentes, eficiência energética, segurança da informação.**

I. Introdução

Através da evolução tecnológica, o surgimento dos Smart Grids (Redes inteligentes) aponta para melhoramentos desde o processo de geração de energia elétrica até o ponto final desta cadeia que é o usuário fim desta energia gerada. Por meio de monitoração, análise, controle e comunicação de todas as partes que integram uma rede elétrica, os Smart Grids buscam alcançar: melhorias no desempenho, otimização do uso dos recursos energéticos, aumento na eficiência energética, alternativas de armazenamento e etc. E, assim garantir um sistema com maior qualidade e eficiência para o consumidor final.

Especificamente na área de medidores de energia elétrica, vem surgindo uma infinidade de novas aplicações, classificadas como Smart Grid, apoiadas basicamente em medidores eletrônicos com leitura do consumo automatizada - AMR (Automatic Meter Reading), ou com leitura de consumo e envio de comandos - AMI (Automatic Meter Infrastructure). A diferença básica entre AMR e AMI, é que a primeira faz uso apenas de uma comunicação unidirecional (para uma simples coleta dos dados de consumo), enquanto a segunda necessita de uma comunicação bidirecional (para executar, por exemplo, um comando de corte no fornecimento de energia a um determinado consumidor)

A infraestrutura de medição avançada (AMI) pode ser contextualizada pela adição de um conjunto de recursos na infraestrutura de medição tradicional, buscando alcançar as seguintes melhorias: do desempenho, da otimização de recursos de energia, da eficiência energética, do armazenamento alternativo, do faturamento automático e remoto e da telemetria. Todas essas características surgem através da introdução dos medidores inteligentes, que são medidores elétricos com software embarcado. Apesar das vantagens tecnológicas na utilização do AMI, é importante caracterizar as ameaças que estes medidores podem trazer. A preocupação sobre a segurança destes equipamentos é um desafio devido ao fato de que os medidores inteligentes operaram num ambiente exposto, sujeitos a acesso não autorizado e assim possibilitando o emprego de manipulação engenharia reversa.

No Brasil, o desenvolvimento e a implementação do AMI possibilitará uma monitoração mais eficiente do consumo de energia elétrica, permitindo um melhor planejamento de expansão da distribuição de energia. Ademais, será um caminho mais efetivo no combate às fraudes de energia elétrica. O mercado brasileiro, atualmente, tem aproximadamente 60 milhões de consumidores, sendo 40% responsável pelo consumo de baixa energia (até 220kWh de consumo mensal). Dentro deste cenário, o Brasil tem experimentado uma situação muito peculiar, no qual as perdas não-técnicas atingem o índice de 20% da energia elétrica produzida, algo em torno de 7,428,000 MWh. Esta energia representa uma perda de quase US$ 1 bilhão por ano.

Após o processo de privatização das companhias de distribuição de energia elétrica nos meados dos anos 1990's, a Agência Nacional de Energia Elétrica determinou que as distribuidoras seriam as responsáveis pelas perdas não-técnicas, forçando estas empresas a investirem em soluções técnicas para reduzir tais perdas. Uma alternativa foi a introdução de medidores AMI dotados de funcionalidades antifraudes. Em adição a essas mudanças, foi necessário, também, criar mecanismo de avaliação destes medidores frente aos requisitos de segurança da informação, com o objetivo de assegurar que as soluções propostas de antifraude e de proteção a engenharia reversa, por exemplo, atendam aos requisitos mínimos de segurança. Esse processo de avaliação dos medidores, no Brasil, é de responsabilidade do Instituto Nacional de Metrologia, Qualidade e Tecnologia (Inmetro).

A figura 1 mostra um exemplo de um sistema de medição centralizada validado e seus componentes. O hub executa as tarefas que se seguem: medição elétrica de energia, a trans-

Este trabalho foi desenvolvido no âmbito do Programa de Pesquisa e Desenvolvimento Tecnológico do Setor de Energia Elétrica regulado pela ANEEL e consta dos Anais do VIII Congresso de Inovação Tecnológica em Energia Elétrica (VII CITENEL), realizado na cidade de Costa do Sauípe/BA, no período de 17 a 19 de agosto de 2015.

C. B. Prado, L. F. R. C. Carmo, D. R. Boccardo, R. C. S. Machado, L. M. Bento, C. F. Novello, A. E. Rincon, L. V. Matieli trabalham no 'Inmetro' (e-mails: cbprado@inmetro.gov.br; lfrust@inmetro.gov.br; drboccardo@inmetro.gov.br;rcmachado@inmetro.gov.br; lmbento-eletrobras@inmetro.gov.br; cfnovello-eletrobras@inmetro.gov.br; aerincon-cgti@inmetro.gov.br; lvmatieli-eletrobras@inmetro.gov.br).

R. A. Oliveira trabalha na Eletrobras Distribuição Rondônia (e-mail: renato.oliveira@eletrobrasrondonia.com).

missão de dados de consumo de energia elétrica para um usuário final (display), e comunicação com a empresa concessionária. Um hub é constituído por um medidor, um relê de corte, uma unidade de comunicação remota (RCU), uma unidade de comunicação local (LCU), e uma unidade central de processamento (CPU). O repetidor contém os mesmos componentes de um hub com a exceção do comando remoto. A CPU representa o núcleo do sistema, sendo responsável para capturar todos os dados dos medidores, executar comandos internos / externos e acompanhar todos os status. O LCU envia informações de consumo por rádio de transmissão a um monitor do usuário final. A RCU inclui um modem que usa transmissões celulares e rádio pacote para lidar com as comunicações bidirecionais entre o hub e a empresa concessionária.

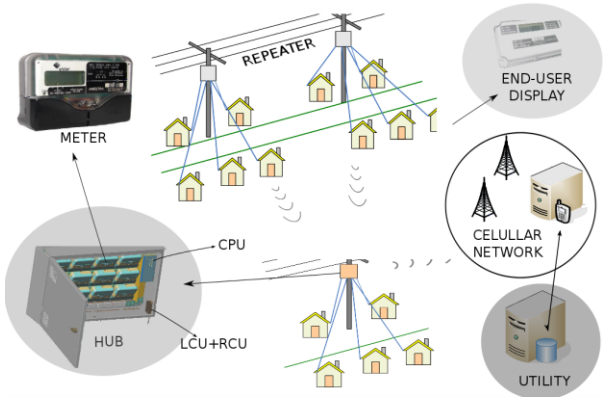


Figura 1. Sistema de Medição Centralizado e seus componentes.

Atualmente, a área de segurança cibernética para Smart Grids vem sendo o foco de pesquisas nesta área [3]. Segurança cibernética, a que nos referimos neste contexto, se refere à aplicação de metodologias que impossibilite (ou torne quase que impossível) que pessoas não autorizadas tenham acesso e poder de manipulação de qualquer serviço ou informação inerente a um Smart Grid e assim utilizá-los para fins inapropriados.

O presente artigo apresenta os resultados do projeto de pesquisa e desenvolvimento, em andamento, código PD-0369-0004/2011, intitulado "Segurança Cibernética em Smart Metering", tendo como proponente a Empresa Eletrobras Distribuição Rondônia e como entidades executoras o Inmetro e o CGTI (Centro de Gestão de Tecnologia e Inovação). Neste trabalho, descreveremos as iniciativas que envolve esse projeto de pesquisa. As seguintes linhas de pesquisa foram desenvolvidas neste projeto:

- Análise de Software – incluindo metodologias para detectar vulnerabilidades e para rastreabilidade de software;
- Proteção de software – com o desenvolvimento de metodologias para verificação de integridade, ofuscação de código e marca d'água dos softwares embarcados;
- Cadeia de confiança de medição – aborda o desenvolvimento uma metodologia de segurança da informação que permita às concessionárias assegurarem a integridade da informação durante toda a cadeia de processamento do Smart Grid. ;
- "Chip" metrológico – visa realizar um estudo de viabilidade, técnico e econômico, de um semicondutor para uso nos medidores inteligentes.

Este artigo segue a seguinte organização: na seção 2 detalharemos cada linha de pesquisa desenvolvida e resultados obtidos; na seção 3 apresentaremos os resultados gerais e na seção 4 as conclusões.

II. Desenvolvimento da Pesquisa

Nesta seção descreveremos cada uma das linhas de pesquisa desenvolvidas no projeto "Segurança Cibernética em Smart Metering" e os resultados obtidos.

A. Análise de Software

Esta linha explora técnicas de análise de código para auditar sua corretude, verificar a inexistência de falhas, verificar se está consistente/coerente com as especificações do projeto, e sua conformidade quanto aos requisitos de segurança.

O processo de auditar um código, no seu formato fonte ou binário, consiste em descobrir vulnerabilidades que os atacantes ou adversários possam explorar, os permitindo identificar brechas de segurança que possam colocar dados sensíveis, serviços ou recursos do negócio em risco.

A vantagem de validar o software por meio do código fonte é que o analisador permite a identificação de estrutura de linguagens de alto nível (laços, procedimentos, classes) de forma direta, possibilitando um melhor entendimento de como o código está estruturado. Contudo, a análise do código fonte tem suas peculiaridades, tais como, a necessidade de confiar no ambiente de compilação e em bibliotecas de terceiros, que podem ser potencialmente vulneráveis, o que não ocorre caso a análise fosse conduzida no código binário.

Existem também outras razões para a análise ser aplicada no nível de código binário, e uma delas seria a análise de binários cuja tabela de símbolos e informações de depuração tenham sido removidas, processo comumente utilizado para proteção de propriedade intelectual. Entretanto, a análise de código binário é muito mais complexa uma vez que os procedimentos não estão claramente delimitados e não existe a distinção entre códigos e dados. Sem informações da tabela de símbolos e sem a tipagem das variáveis armazenadas e manipuladas entre memória e registradores, fica difícil identificá-las e distinguí-las de números inteiros ou endereços de memória.

O Inmetro, por meio da Portaria 11/2009, exige a abertura do software (código fonte), decisão essa resultante de uma ampla discussão interna e externa com fabricantes, influenciada pela complexidade arquitetural dos sistemas atuais. Outros regulamentos também estão solicitando analogamente a abertura do código fonte para análise de segurança. No entanto, um problema surge com esta demanda: como garan-

tir que o código binário a ser utilizado pelo sistema foi efetivamente gerado a partir do código fonte disponibilizado pelo fabricante e avaliado pelo Inmetro?

Este problema é chamado de rastreabilidade de software e tem por objetivo saber se o comportamento descrito em um determinado código fonte (escrito em alguma linguagem de programação) está refletido em um código binário apresentado como sendo o correspondente compilado àquele código fonte. Duas abordagens imediatas são possíveis, mas não práticas. A primeira consiste em auditar o processo de compilação no ambiente de desenvolvimento do fabricante. O problema desta abordagem refere-se à dificuldade de se conduzir uma auditoria perfeita no ambiente de compilação do fabricante. A segunda abordagem consiste em replicar o ambiente de desenvolvimento usado pelo fabricante (compilador, ligador) para possibilitar uma recompilação do código, seguida da comparação das linguagens. As desvantagens desta abordagem são: custo para reproduzir o ambiente do fabricante (compra de ferramentas, máquinas, compiladores), e a grande possibilidade em produzir um código binário diferente daquele apresentado pelo fabricante devido a detalhes de configuração, como por exemplo, uma otimização aplicada pelo compilador ou uma versão ligeiramente diferente de alguma biblioteca.

Para auxiliar na eficácia da análise de código, é usada a ferramenta Crystal Revs. Ela nos permite criar diagramas e relatórios automáticos, úteis tanto para o contato inicial com o código quanto para consulta durante toda a análise, assim como grafos de fluxo de controle ou grafos de chamadas de função e bibliotecas.

Com o Crystal Revs, é possível avaliar se existem variáveis, funções ou bibliotecas declaradas e que não são usadas. Também é possível identificar bibliotecas ou funções com muitas chamadas, e assim, identificar pontos candidatos para a análise. Vulnerabilidades nesses pontos candidatos podem comprometer diversos módulos do sistema.

A ferramenta apresenta as seguintes três estratégias de análise: compreensão, pontos candidatos e generalização.

A estratégia de compreensão envolve a leitura do código com direção para frente. As opções do auditor são: o ponto inicial, o escopo e a sensibilidade da leitura. Os objetivos da estratégia de compreensão são identificar e analisar:

- Inicialização, funcionalidades, e término do programa;
- Tipos de acesso e entradas do usuário;
- Módulos, classes, algoritmos e variáveis relevantes.

O conjunto de instruções que realiza uma preparação para a execução das funcionalidades compõe a inicialização do software. Nesta etapa, podem ocorrer procedimentos como desabilitar interrupções, verificar a integridade do software, habilitar ou desabilitar LEDs, etc. Em geral, os softwares de medidores possuem um loop principal, que caracteriza a espera por interrupções ou comandos de usuário para a execução de suas funcionalidades.

O ponto de término do programa é aquele em que não é mais possível voltar a executar as funcionalidades. A análise deve responder a algumas questões, como por exemplo: (a) existe um comando de término? (b) quais são os procedimentos para finalizar o programa? (c) Há algo sobre algum ponto de saída “anormal”? (d) O término do programa sem os procedimentos determinados pode comprometer os dados ou causar problemas na próxima execução?

Inicialmente, pode-se fazer uma análise da função principal do sistema, para identificar comandos de inicialização, funcionalidades e término, e familiarizar-se com a estrutura do programa. É possível relacionar as funcionalidades principais com as chamadas de função do programa e grafos de fluxo de execução de cada função. Com isso é possível ver detalhes de função como declaração de variáveis, uso de código de instruções e funcionalidades que auxiliam a análise do código.

A estratégia de pontos candidatos utiliza métodos de identificação de padrões de código que caracterizem erros comuns e/ou possíveis vulnerabilidades. Para compreender o impacto de alguma vulnerabilidade encontrada pela ferramenta, realiza-se uma leitura para trás, em busca, por exemplo, do ponto de entrada de dados pelo usuário. Ferramentas de análise estática do código podem ser utilizadas para a identificação destes padrões.

MISRA, CWE, CERT, entre outros, são conjuntos de boas práticas e iniciativas de código seguro. O Crystal Revs possui a ferramenta de avaliação para os padrões MISRA 1998 e MISRA 2004. Com isso, é possível gerar um relatório configurando quais regras serão aplicadas na avaliação automática do código.

A estratégia de generalização foca na estrutura lógica e arquitetura do programa, relacionando o comportamento real do sistema com o descrito na documentação de apresentação das funcionalidades do sistema. Pode começar da documentação para o código ou vice-versa. É uma leitura insensível em geral, mas pode ser sensível ao fluxo se for necessário para entender a implementação.

A estratégia de generalização pode começar com a leitura de um grafo de fluxo, em menor detalhe. Se houver necessidade de investigar em maior detalhe, as estratégias de compreensão podem ser utilizadas, como por exemplo, a visualização de um nó de alto nível para exibir as instruções omitidas.

Um dos relatórios gerado pelo Crystal Revs possui métricas que representam medidas de complexidade e volume, que podem ser comparadas a outros softwares avaliados, para uma ideia de dimensões. Para a avaliação da estrutura do programa e funções, uma leitura mais detalhada dos diagramas do Crystal Revs com escopo de projeto e função pode auxiliar a análise de código.

No escopo deste projeto, esta ferramenta foi otimizada para realizar uma análise do software embarcado com foco principal em 6 regiões de interesse do código fonte:

- Funcionalidade principal
- Verificação de integridade
- Controle de acesso
- Carga de software
- Proteção de dados sensíveis
- Verificação de funções não-utilizadas

Por meio do relatório HTML do "Crystal Revs" é possível identificar o ponto de início de execução, que irá chamar todas estas funcionalidades. Desta forma, é possível rastrear

procedimentos de inicialização, comandos de interação com o usuário, exibição das interfaces, etc. Assim, possíveis vulnerabilidades podem ser detectadas.

Uma outra importante aplicação do "Crystal Revs" na análise do software embarcado nos medidores inteligentes é a que se refere à verificação de funções não-utilizadas. É possível gerar um relatório identificando o arquivo e a linha da função não-utilizada. Em alguns casos, podem ser encontrados arquivos em que todas as suas funções nunca são chamadas, ou seja, podem não estar sendo utilizadas. Assim, é possível restringir a análise apenas ao código que está sendo realmente utilizado, otimizando o tempo da avaliação.

Foi adquirido durante o projeto uma ferramenta similar ao Crystal Revs, a LDRA Tools, porém com maior capacidade de análise do software. As diferenças básicas são:

- Ferramenta automatizada, enquanto o Crystal Revs é manual;
- Além das funcionalidades do Crystal Revs, a LDRA Tools agrega a funcionalidade de análise de vulnerabilidades do código seguindo o padrão CERT;
- Realiza a análise dinâmica do software embarcado, ou seja analisar o comportamento durante a sua execução;

Ambas as ferramentas, Crystal Revs e LDRA Tools, são atualmente utilizadas no processo de análise do software embarcado dos medidores inteligentes.

B. Proteção de Software

Esta linha de pesquisa explora técnicas de transformação de código para proteção de software. Uma técnica de proteção de software pode ser definida como um conjunto de procedimentos para dificultar um atacante ou adversário de obter ganhos sobre o software, seja por motivos financeiros, sabotagem ou vingança. O conhecimento destas técnicas exerce maior impacto em cenários que podem afetar infraestruturas nacionais caso atacadas, como os smart grids do setor elétrico. Um cenário de ataque em um smart grid envolve: a captura de um instrumento de medição disposto em um ambiente fisicamente acessível e a sua engenharia reversa a fim de ter posse de informações secretas (chaves criptográficas) que venham a servir para propagar um ataque. Um exemplo de ataque seria o envio de informações errôneas para o sistema de controle central podendo causar um blackout em um segmento da rede elétrica.

Avanços em análise de programas e tecnologias de engenharia de software têm levado ao aperfeiçoamento das ferramentas para análise e desenvolvimento seguro de software. Estas fornecem subsídios para que os softwares sejam mais eficientes, seguros e protegidos, o quanto possível, de vulnerabilidades. No entanto, estes mesmos avanços aumentam a capacidade para a engenharia reversa com o objetivo de descobrir vulnerabilidades ou realizar alterações indevidas. Por exemplo, para que um atacante ou adversário mal intencionado explore uma vulnerabilidade em um sistema, o mesmo tem que primeiro identificá-la. Analogamente, para alterar o código embarcado de um dispositivo, o atacante tem primeiro que analisar a forma como adulterar o código sem que a funcionalidade do sistema seja afetada e que nenhum mecanismo de segurança seja ativado.

As técnicas de proteção de software visam dificultar a engenharia reversa do software embarcado, assegurar que ele execute como esperado e rastreá-lo diante de uma possível distribuição ilegal [4]. As técnicas de proteção de software que podem ser utilizadas para esta finalidade são: ofuscação de código, incorruptibilidade e marca d'água. Técnicas de ofuscação de código dificultam engenharia reversa através de um aumento no grau de ininteligibilidade do código. Este aumento se caracteriza por modificações sintáticas que dificultam a compreensão do software, mantendo, porém, as funcionalidades originais do programa, ou seja, sua semântica. Técnicas de incorruptibilidade asseguram que o software execute como esperado, mesmo na tentativa de modificação ou monitoração por um atacante ou adversário. E por fim, técnicas de marca d'água referem-se ao ato de embarcar uma informação num objeto com o objetivo de torná-lo posteriormente rastreável.

Assim, esta linha de pesquisa visa a investigar o uso das técnicas de proteção de software a serem utilizadas nos medidores inteligentes. Uma vez que estes possuem características bem específicas e na maioria das vezes bem peculiares, o balanceamento adequado entre proteção e impacto no desempenho no uso destas técnicas nestes medidores, apontam o caminho para assegurar a privacidade, incorruptibilidade e a rastreabilidade dos softwares embarcados.

Ofuscação:

A aplicação de ofuscação de código em smart grids refere-se à necessidade da privacidade de constantes e dados, confidencialidade do código para evitar a descoberta de vulnerabilidades e que possam comprometer a integridade do código. Por exemplo, caso chaves criptográficas utilizadas para garantir a privacidade dos dados entre dispositivos comunicantes fossem reveladas por um atacante ou adversário, o mesmo poderia desvendar segredos da aplicação, como por exemplo, capturar comandos disponíveis, possibilitando o envio de comandos do sistema, por exemplo de suspensão do fornecimento de energia, sem ter as credenciais necessárias. Dados de medição e constantes de calibração também ora localizados, estão sujeitos a manipulação por um indivíduo mal intencionado para fins inapropriados. Vulnerabilidades presentes no código ora localizadas podem ser exploradas. A ofuscação de código também está relacionada com a garantia da integridade de softwares embarcados nos medidores inteligentes, uma vez que um software de difícil discernimento é de difícil modificação.

A ofuscação de código é uma técnica de proteção de software que visa dificultar a engenharia reversa a fim de impedir o comprometimento da propriedade intelectual e a extração de dados sensíveis de um programa. A engenharia reversa é um processo que tenta reconstruir o código executável de um programa para uma linguagem de mais alto nível com o intuito de fornecer informações desse programa. A engenharia reversa é comumente dividida em duas fases: desmontagem (disassembly) e descompilação. Na fase de des-

montagem, o código binário é traduzido para código assembly, ou seja, uma representação dos códigos de máquina que devem ser executadas em uma determinada arquitetura de processamento. Na fase de descompilação, o código assembly, gerado pela fase anterior, é utilizado para gerar estruturas de mais alto nível que facilitam a análise e extração de conhecimento do programa.

As técnicas de ofuscação transformam o código de um programa a fim de produzir um programa mais difícil de ser entendido, mas que possui o mesmo comportamento do programa original. A ofuscação de código é aplicada para comprometer o processo de engenharia reversa, podendo comprometer tanto a fase de desmontagem como a fase de descompilação das ferramentas utilizadas para fazer engenharia reversa. Em alguns casos, a ofuscação também é utilizada para produzir diferentes versões de um programa. Isso é feito a fim de combater ataques de colisão, no qual um atacante com posse de mais de uma cópia do programa, consegue descobrir a localização de uma determinada propriedade do programa através da comparação das cópias, como por exemplo, um atacante realizar um ataque de colisão a fim de identificar o número serial de um programa. Dessa forma, o uso da ofuscação de código pode dificultar um ataque de colisão, pois ao usar diferentes técnicas de ofuscação para produzir diferentes versões do mesmo programa dificultará a identificação de padrões entre as diferentes versões do programa, já que as cópias serão diferentes em sua forma [5].

A maioria das técnicas de ofuscação atua de forma a comprometer a fase de descompilação das ferramentas de engenharia reversa, [6], [7], [8]. Nesse contexto, existem técnicas que reordenam ou incluem blocos básicos (vértices) e relacionamentos entre esses blocos (arestas), dificultando a análise do Grafo de Fluxo de Controle (CFG). Outras técnicas podem dificultar a análise do fluxo de dados. Por exemplo, podemos citar as técnicas que substituem expressões simples do programa por outras que sejam mais complexas, ou seja, simples atribuições podem ser substituídas por outras contendo expressões matemáticas complicadas [1]. Além das técnicas que comprometem a descompilação, existem técnicas que induzem os disassemblers a traduzirem incorretamente as instruções de um programa, ou seja, comprometem a fase de desmontagem das ferramentas de engenharia reversa. Tais técnicas baseiam-se na fragilidade das convenções de compilação, ou seja, nos pressupostos assumidos pelos disassemblers durante a tradução do código executável. Exemplos de convenções de compilação são: a sequência de instruções em um procedimento de entrada (prólogo), em uma saída de procedimento (epílogo) e para realizar chamadas e retornos de funções. Tais convenções não são consideradas padrões, pois são específicos para cada compilador e, mesmo para um determinado compilador, podem variar de acordo com o esquema de otimização selecionado. Dessa forma, quando tais convenções não são seguidas, seja por conta da otimização ou ofuscação de um programa, a desmontagem do mesmo pode ser questionável e com isso quaisquer análises subsequentes podem estar incorretas. Dentre as técnicas capazes de comprometer a desmontagem destacamos a ofuscação de chamada, ofuscação de retorno [9] e a ofuscação de falso retorno [10].

Assim como as técnicas de junção e separação de funções , as ofuscações de chamada, de retorno [9] e de falso retorno [10] comprometem a delimitação das funções de um programa. As ofuscações de chamada e de retorno modificam a sequência de instruções para realizar uma chamada e o retorno de uma função, respectivamente. Enquanto a ofuscação de falso retorno insere uma instrução de retorno no meio de uma função. Além de comprometer a delimitação de funções, tais ofuscações permitem alterar o fluxo de controle do programa, o que pode induzir a tradução incorreta desse programa já que os algoritmos utilizados pelos disassemblers não sabem identificar corretamente os destinos das instruções que redirecionam o fluxo de controle.

Uma vantagem das técnicas de ofuscação que compromete a desmontagem é que é difícil desenvolver ferramentas capazes de reverter esse tipo de ofuscação (desofuscadores), pois é difícil identificar os pontos do programa que foram ofuscados já que as instruções utilizadas pela ofuscação podem ser dispostas de forma aleatória no programa. Outra vantagem é que esse tipo de ofuscação não causa um impacto muito negativo sobre recursos computacionais (processamento e memória) do dispositivo onde o programa ofuscado está executando.

Na prática, as transformações realizadas por uma técnica de ofuscação simplesmente substitui, insere, remove e reordena instruções em um programa. Tais transformações podem ser implementadas em pelo menos três níveis do programa: (i) alto, (ii) intermediário e (iii) baixo. No nível alto, essas transformações são realizadas no código fonte, ou seja, no arquivo texto contendo os algoritmos escritos usando uma linguagem de alto nível (C, Java, Python, etc.). No nível intermediário, essas transformações são realizadas no código assembly e, por ultimo, no nível baixo, essas transformações são realizadas diretamente no código executável do programa, ou seja, modifica os códigos de máquina que representam as instruções que devem ser executadas.

Escolher em que nível aplicar as transformações no programa está diretamente relacionado ao custo-benefício entre a dificuldade de realizar essas transformações e o ganho da generalidade que essas transformações podem fornecer. Por exemplo, realizar transformações em um código assembly abrange mais programas do que realizar transformações em um nível, já que em um determinado momento os códigos dos programas escritos com linguagens de alto nível irão ser transformados em código assembly. Entretanto, implementar as transformações diretamente em um código assembly é mais difícil do que implementar essas transformações em um código escrito usando uma linguagem de alto nível, já que será necessário conhecer as especificidades da arquitetura de processamento onde esse programa será executado. Dessa forma, quanto mais baixo for o nível de implementação, maior a generalidade da transformação e vice-versa.

Um dos desafios para utilizar técnicas de ofuscação no cenário desse projeto, medidores inteligentes, é em relação à heterogeneidade de arquiteturas utilizadas visto que cada fabricante desenvolve o seu medidor utilizando uma arquitetura específica, como por exemplo, ARM, AVR e etc. Isso

nos leva a diferentes contextos em que se deseja aplicar as técnicas de ofuscação, o que dificulta o processo de automatização da aplicação de tais técnicas, pois é necessário considerar as particularidades de cada arquitetura durante a implementação dessas técnicas (vide figura 2). Dessa forma, com o intuito de minimizar o esforço para a criação de ferramentas capazes de aplicar automaticamente as técnicas de ofuscação para diferentes arquiteturas e que abranja o maior número possível de programas, estamos desenvolvendo uma API (Application Programming Interface) para a linguagem de programação C/C++ que suporte o desenvolvimento de ferramentas capazes de realizar transformações em um código assembly a fim de implementar técnicas de ofuscação para diferentes arquiteturas, evitando assim o retrabalho do desenvolvedor em implementar as mesmas técnicas para arquiteturas distintas [11]. Por meio do uso desta API, um desenvolvedor será capaz de implementar de forma genérica uma determinada técnica de ofuscação, enquanto a API se encarrega de efetuar as rotinas de inserção, substituição, reordenação e remoção de instruções associadas de acordo com a arquitetura que será utilizada.

A API foi dividida em três camadas que cumprem para a transformação de um código assembly, como mostrado na figura 2: (i) API pública: é a camada mais externa acessível ao desenvolvedor. Fornece um conjunto de operações genéricas de inserção, substituição, reordenação e deleção que serão utilizadas para manipular o código assembly (ii) Drivers: unívocos para cada arquitetura e desenvolvidos utilizando a linguagem XML, definem as operações genéricas e suas correspondentes na arquitetura específica. O mesmo conjunto de instruções genéricas é utilizado para todas as arquiteturas. (iii) Métodos específicos: Esta é a camada que de fato executa todas as operações. É responsável por manipular diretamente o código assembly, gerando um novo arquivo com as modificações.

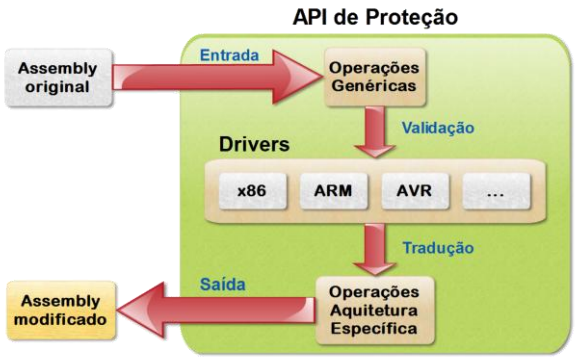


Figura 2. Visão interna da API mostrando as diferentes etapas de seu funcionamento.

Além do desenvolvimento da API de transformação de código, nossos esforços estiveram focados em outra linha de trabalho, mais especificamente, no desenvolvimento de uma técnica de ofuscação capaz de camuflar dados do programa em meio ao código do mesmo, tornando difícil a localização e identificação desses dados [12]. Tal abordagem é baseada em software com o intuito de adicionar camadas de dificuldade para o atacante que esteja realizando a engenharia reversa de um programa. Dessa forma, a técnica de ofuscação desenvolvida pode coexistir com outras técnicas, inclusive com aquelas baseadas em hardware, aumentando ainda mais o grau de proteção de um programa.

Um programa comumente é dividido em dois segmentos. Um segmento de código, contendo as instruções do programa e um segmento de dados, contendo as constantes e variáveis utilizadas pelo programa. A ideia da técnica de ofuscação proposta é proteger os dados sensíveis de um programa localizado no segmento de dados. Para isso essa técnica de ofuscação combina técnicas de ofuscação existentes na literatura (ofuscação de chamada, ofuscação de falso retorno e ofuscação de retorno) com a manipulação do fluxo de controle do programa com o intuito de criar trechos no segmento de código, capazes de abrigar dados. Dessa forma, ao armazenar dados no segmento de código, as ferramentas de engenharia reversa serão induzidas a traduzir incorretamente esses dados, como se fossem instruções do programa.

Assim, trabalhamos em técnicas de ofuscação que dificultam ataques recorrentes já que as propostas atuais geralmente tratam a proteção de software de forma estática, ou seja, empregam técnicas de ofuscação antes do programa estar em execução, permanecendo intacto até que seja corrompido por um atacante. Dessa forma, após realizar a engenharia reversa e identificar uma determinada vulnerabilidade e a localização de um dado sensível, um atacante pode ser capaz de criar uma ferramenta que de forma automatizada pode comprometer uma mesma instância repetidas vezes em tempos diferentes ou comprometer um sistema como um todo, caso o cenário de ataque seja um ambiente onde haja várias cópias exatas do mesmo programa, permitindo após analisar uma única instância, comprometer todas as cópias. Dessa forma, direcionamos nossos esforços em um esquema de proteção de software dinâmico que faz com que o software embarcado nos medidores mude sua forma periodicamente em tempo de execução. Atualmente estamos trabalhando num esquema de proteção de dados dinâmico. A ideia é baseada na proposta anterior [12] que visa camuflar dados sensíveis. Neste novo esquema, propomos que os dead execution spots sejam criados dinamicamente e que os dados sensíveis sejam movidos entre eles em tempo de execução. Esse novo esquema visa dificultar ataques recorrentes e massivos já que o código do programa muda em tempo de execução. Outra vantagem dessa proposta é que é possível prover diversidade de software de forma dinâmica, facilitando a criação de cópias distintas de um mesmo programa já que o esquema proposto indica que um programa a ofuscação dinâmica é desencadeada por uma característica única de cada instância de um sistema embarcado.

Marca D'água:

Técnicas de marca d'água/fingerprinting realizam a inser-

ção de uma informação de identificação em programas de computador, permitindo reivindicações posteriores de autoria/propriedade e, portanto, desencorajando a cópia ilegal dos mesmos. Considerando o modelo de validação de medidores de energia elétrica, e utilizando o conceito de fingerprinting, propomos um protocolo de segurança que permite identificar responsáveis por possíveis "vazamentos de código" numa eventual terceirização da avaliação de código por parte da Autoridade Metrológica [13].

Neste protocolo, a Autoridade Metrológica gera um hash do código a ser avaliado acrescido da informação de identificação do avaliador, assina esse hash, e o conjunto hash assinado + identificação do avaliador compõem a informação *w* a ser embarcada. Em seguida, a Autoridade Metrológica codifica *w* na forma de um grafo apropriado, embarcando-o no programa original *P* (em uma região secreta *k*).

Como em todo cenário de segurança, fingerprints estão sujeitos a certos ataques, tais como: adição, no qual o atacante inclui um outro fingerprint no programa; distorção, em que o atacante altera o programa sintaticamente por meio de transformações que preservam a semântica do programa original, mas que podem incapacitar a recuperação do fingerprint; e subtração, onde o atacante descobre a localização do fingerprint e simplesmente o remove.

Analisando o protocolo proposto diante destes ataques, pode-se observar que para o ataque de adição, a necessidade de assinatura pela Autoridade Metrológica impede que o avaliador malicioso insira no código do programa um outro fingerprint, pois o mesmo não possui a chave privada da Autoridade Metrológica. A análise dos ataques de distorção e subtração dependem do esquema de codificação de marca d'água utilizado. Inicialmente, consideramos o uso do esquema de codificação de marca d'água proposto por Chroni e Nikolopoulos [14,15] e fizemos um amplo estudo para compreender o comportamento das marcas d'água obtidas por tal esquema diante de ataques de distorção. Identificamos que as marcas d'água geradas pelo esquema de Chroni e Nikolopoulos possuem redundância suficiente para se recuperar de ataques de distorção que realizem até cinco operações de inserção e/ou remoção de arestas do grafo que codifica o fingerprint [16]. Além disso, para os ataques de distorção que causam a remoção de 3 ou mais arestas do grafo, apresentamos algoritmos polinomiais que recuperam, sempre que possível, as arestas removidas [17].

Durante o desenvolvimento destes trabalhos, ao considerarmos ataques de subtração, observamos que a elaboração de marcas d'água eficientes está relacionada, não apenas às características do esquema proposto, mas principalmente em como estão relacionadas com o grafo de fluxo de controle do programa no qual a marca d'água será embarcada.

Existem na literatura diversos trabalhos que discutem técnicas para elaboração de programas, dentre elas podemos destacar a programação estruturada, que consiste num conjunto de padrões de qualidade que torna o programa mais detalhado, legível, confiável e de fácil manutenção, utilizando essencialmente as instruções "se-então-senão" e "enquanto-faça" [18]. Como o princípio básico da programação estruturada é que um programa é composto por blocos elementares de código que se interligam através de três mecanismos básicos, que são sequência, seleção e iteração, os programas assim descritos são de fácil entendimento e apresentam um melhor controle sobre o fluxo de execução do código, além de um melhor desempenho. Por isso, essa técnica vem sendo amplamente utilizada ao longo do tempo, em especial na elaboração dos programas para controle metrológico embarcados nos instrumentos de medição.

Diante deste cenário, estamos trabalhando na caracterização dos grafos de fluxo de controle de programas estruturados (programas sem "goto") [19], e, posteriormente, pretendemos propor um esquema de marca d'água no qual o grafo obtido pertença a esta classe de grafos. Com o grafo gerado por esse novo esquema pertencendo à mesma classe do grafo de fluxo de controle do programa no qual será inserida, a marca d'água ficará "invisível" para o agente mal intencionado, dificultando os diversos tipos de ataques. Numa versão preliminar deste esquema de marca d'água, apresentamos um algoritmo de codificação capaz de gerar mais de um grafo marca d'água para qualquer identificador com o tamanho do grafo muito próximo a representação binária do identificador, além de possibilitar o uso de códigos de detecção e correção de erros para prover resistência a ataques de distorção [20].

Verificação de integridade:

A verificação de integridade de software refere-se ao processo de verificar se o software em execução em um determinado medidor inteligente é exatamente o mesmo que foi previamente aprovado pela autoridade competente. Dentre os vários métodos de verificação de integridade de software, o mais simples e fácil de conduzir é aquele que tem acesso direto ao código em execução no dispositivo - ou seja, é possível realizar o download completo da área de memória onde o programa em execução é armazenado. No entanto, apresenta a desvantagem de que o código de execução permanece disponível para qualquer pessoa que tem acesso ao dispositivo. Tal aspecto compromete a propriedade intelectual do desenvolvedor de software e pode reduzir a segurança do dispositivo. Além disso, o processo de auditoria pode ser inconveniente, uma vez que o auditor deve ter acesso direto ao chip que armazena esse programa - isso nem sempre é prático ou mesmo possível. Uma abordagem mais sofisticada é o uso de hardware especial que oferece uma funcionalidade de retornar um resumo criptográfico (muitas vezes através da aplicação de uma função hash) do código de programa em execução no "chip." Esta funcionalidade protege o conteúdo e, portanto, garante a proteção da propriedade intelectual do fabricante. Por outro lado, há ainda a necessidade de ter acesso direto ao chip, além de que chips com esta funcionalidade são caros e difíceis de serem achados no mercado.

Recentemente, uma abordagem alternativa vem sendo investigada. Em tal abordagem, baseada no conceito de introspecção [21], uma série de comandos de verificação é enviada para o programa em verificação, permitindo que uma

pessoa autorizada possa verificar a integridade do software pelo seu comportamento em resposta a esses comandos. As vantagens de tal abordagem são evidentes: além do código de programa permanecer protegido contra o acesso não-autorizado, é possível enviar os comandos de verificação através da interface de comunicação do, não sendo necessário retirar o chip em que o programa está armazenado.

Mais especificamente, a idéia de verificação de software baseado em introspecção é solicitar que um programa retorne mensagens que são calculados com base no seu próprio código. Na prática, a única necessidade da técnica de introspecção é que o dispositivo que executa a rotina de verificação tenha instruções que permitem o acesso ao seu próprio código de memória. Esta abordagem apresenta bons resultados quando se assume que não existe memória livre e que o código não pode ser comprimido. Se esses pressupostos não são válidos (ver, por exemplo, [22]) - é fácil para um software malicioso manter uma cópia do software original e realizar as rotinas de verificação de integridade sobre a cópia do código original(figura 3).

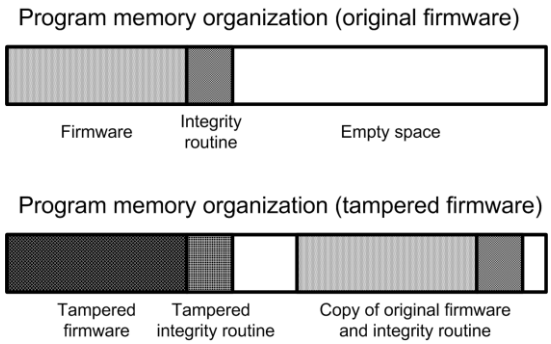


Figura 3. Firmware corrompido (ataque de replicação da memória).

Nós propusemos um método de verificação de integridade de software que solicita ao programa em execução que retorne os chamados "Message Authentication Codes" (MAC) que dependem de seu próprio código. Um algoritmo de MAC é uma função cuja entrada é constituída de uma mensagem de comprimento arbitrário e uma chave secreta, produzindo um resultado de comprimento fixo, chamado de autenticação, como saída. A figura 4 mostra os números de 1 a 5 representando a sequência de desafio-resposta para o procedimento de verificação da integridade.

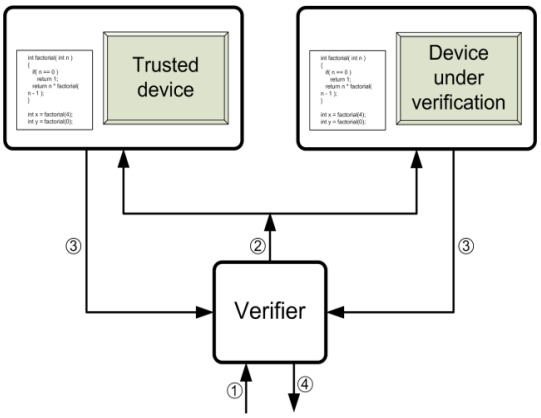


Figura 4. Método de Verificação de Integridade. Os números representam a sequência de desafio-resposta.

A técnica anteriormente descrita não só proporciona a verificação da integridade do software - o que é conseguido através da dependência do comportamento do software para o seu próprio código - mas também proteção intelectual do software - o que é conseguido pelo fato de não ser possível obter o código do programa a partir de um conjunto de respostas usando MAC. A principal razão para a utilização de operações criptográficas que são aplicadas sobre o código de programa é exatamente a proteção deste código.

Uma vez que o software em verificação não pode adivinhar a priori qual será a pergunta a ser feita, poderia haver apenas duas possíveis estratégias para que um software malicioso pudesse "enganar" o processo de verificação. Uma primeira estratégia possível seria, então, armazenar cada resposta possível no software malicioso e simplesmente devolver a resposta desejada. Tal abordagem é claramente impraticável, contanto que qualquer padrão MAC tem um espaço de chave em torno de 2128 chaves.

A segunda estratégia que um software mal-intencionado pode usar para enganar o processo de verificação seria a de manter uma "cópia" do programa original e realizar todos os cálculos do MAC sobre essa cópia (em vez do programa modificado) retornando, em seguida, a resposta esperada. Para se combater essa estratégia, devem ser tomados alguns cuidados adicionais. A primeira e mais simples medida preventiva é garantir que nenhum espaço de memória não utilizado seja deixado livre. Isso pode ser feito facilmente, basta preencher a memória do programa não utilizada com uma sequência aleatória de bits.

Foi desenvolvido uma ferramenta de verificação de integridade de software (SIVT), que emprega a técnica de introspecção proposto. O SIVT fornece uma interface única para um operador responsável por executar a verificação de integridade dos softwares embarcados nos medidores de energia elétrica em campo. Esta ferramenta também gerencia a coexistência de diferentes versões de software, cada um contendo o seu identificador único.

Esta abordagem, atualmente, já vem sendo utilizada na verificação integridade dos medidores inteligentes aprovados pelo Inmetro.

C. Cadeias de Confiança dos Medidores

As atividades desta linha de pesquisa concentraram-se no estudo e no desenvolvimento de modelos de segurança e de protocolos e algoritmos criptográficos visando serviços de segurança específicos ao cenário de Smart Grids. Em particular, as seguintes linhas de pesquisa foram conduzidas.

1) Estudo de protocolos de "fair non-repudiation exchange".

Irrefutabilidade refere-se à capacidade de tornar inegável que um indivíduo executou uma ação ou tomou conhecimento de uma informação. *Troca justa* refere-se à capacidade de efetuar uma troca de informações em uma "etapa única", ou seja, sem que uma das partes receba sua informação antes da outra parte. Protocolos de "fair non-repudiation exchange" têm por objetivo proporcionar a troca justa de informações de irrefutabilidade, tais como assinaturas digitais, e são essenciais em cenários em que dois usuários mutuamente não-confiáveis precisam registrar acordos acerca de determinada informação (por exemplo, assinatura digital de um contrato). Adicionalmente, o estudo de protocolos de fair non-repudiation exchange provou ser um estágio ao desenvolvimento de protocolos envolvendo fingerprinting com o objetivo de registrar a entrega de objetos digitais (por exemplo, software) a terceiros, com o objetivo de ter uma avaliação do software por este terceiro.

2) Desenvolvimento de protocolos de fingerprinting para delegação de análise de software.

Fingerprinting refere-se ao processo de identificar unicamente um objeto. Técnicas de fingerprinting aplicam-se a objetos digitais, tais como software, permitindo rastrear um aplicativo de software que foi enviado a terceiros (por exemplo, para avaliação), conforme detalhamos na seção anterior. O problema é que os protocolos clássicos não oferecem a capacidade de irrefutabilidade, uma vez que aquele que entrega o software é o responsável por inserir a informação de fingerprinting. Ao longo do projeto, especificamos um protocolo de fingerprinting baseado no conceito de oblivious transfer e que é capaz de proporcionar irrefutabilidade, uma vez que nem mesmo a parte que entrega o software (contendo fingerprinting) é capaz de saber o conteúdo exato do software que foi entregue (o conceito é altamente contra-intuitivo - para mais informações sugerimos consultar a literatura sobre o *oblivious transfer*).

3) Protocolo de marcas d'água baseado em provas de conhecimento nulo.

Conforme descrevemos na seção anterior, marcas d'água (digitais) referem-se a informações embarcadas em objetos (digitais) e que podem ser recuperadas mesmo após modificações executadas sobre aquele objeto. No contexto de aplicativos de software, marcas d'água podem ser utilizadas para embarcar informações de autoria ou propriedade, ou ainda identificar unicamente o indivíduo autorizado a utilizar tal aplicativo. Idealmente, uma marca d'água é um objeto de difícil remoção. No entanto ao exibir uma marca d'água - por exemplo, para comprovar a autoria de um software perante um juiz - a sua remoção passa a ser facilmente executável por eventuais atacantes. Ao longo do projeto, foi possível desenvolver um algoritmo que permite embarcar uma marca d'água em um objeto digital qualquer e demonstrar a sua existência sem que seja necessário apresentar a sua localização, dentro do objeto. O protocolo desenvolvido permite que se exiba a informação de uma marca d'água sem que isto torne mais fácil, a um atacante, removê-la. (Mais uma vez, o resultado é contra-intuitivo - sugerimos consultar a literatura sobre *zero-knowledge proofs*).

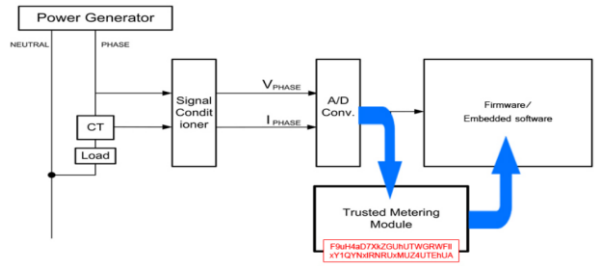
4) Infraestrutura de chaves públicas para aplicações metrológicas

Assinaturas digitais são a solução-padrão para prover autenticidade e irrefutabilidade a uma informação. No entanto, para atestar que determinada informação realmente teve origem em um determinado indivíduo, é imprescindível dispor de uma fonte confiável que informe a chave pública daquele indivíduo. Uma infraestrutura de chaves públicas é uma entidade responsável por fazer uma associação entre identidades e chaves públicas correspondentes. Com o uso de protocolos baseados em assinatura digital no contexto de Smart Grids, torna-se necessário identificar quais são os requisitos para uma infraestrutura de chaves públicas aplicada a este cenário. Ao longo do projeto, investigamos os requisitos para infraestruturas de chaves públicas com foco na medição inteligente, ou seja, em que cada medidor possuiria a sua chave pública, e uma terceira parte confiável seria responsável por certificar medidores, ou seja, informar qual chave pública estaria associada a qual medidor. Identificamos diversos modelos de certificação digital e definimos os requisitos de segurança para os vários elementos envolvidos com a infraestrutura de chaves públicas - desde o medidor, responsável pela proteção de sua própria chave privada, até as autoridades certificadoras, responsáveis pela emissão de certificados.

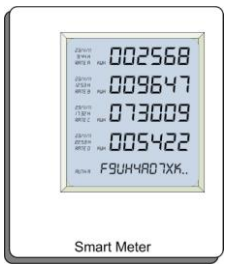
5) Algoritmos e protocolos seguros para tarifação por postos horários

A iminência do modelo de tarifação por postos horários traz novos e interessantes desafios no contexto da segurança da informação. Isto porque a informação metrologicamente relevante deixa de ser apenas o "consumo acumulado" e passa a ser o "consumo acumulado por posto tarifário". Além disso, o "posto tarifário" não é uma entidade fixa, podendo ser modificado ao longo do tempo, de acordo com os "estímulos" que o regulador deseje dar ao consumidor de energia. Ao longo do projeto, foi possível desenvolver modelos de segurança da informação de consumo de energia por postos tarifários, inclusive com o desenvolvimento de um modelo de integridade das informações por postos tarifários baseado em assinatura digital [23]. A figura 5.a demonstra um mecanismo baseado em assinatura de digital no qual o módulo criptográfico atua como raiz de confiança para a cadeia legalmente relevante do medidor inteligente. A figura 5.b. representa um display no qual se pode verificar as in-

formações de consumo com sua assinatura digital.



(a)



(b)

Figura 5. (a) Modelo de raiz de confiança de um Medidor Inteligente. (b) Autenticador de consumo visualizado em display.

D. "Chip" Metrológico

Esta frente refere-se ao desenvolvimento de uma arquitetura computacional que facilite o atendimento a requisitos de segurança da informação em medidores especificados em regulamentos e portarias do Inmetro (rastreadas em normas internacionais). Tal arquitetura terá sua organização baseada na presença de um dispositivo eletrônico "chip" cuja especificação será um dos produtos finais do presente projeto. Este "chip" funcionaria como um controlador metrológico customizado através do qual seja possível: realizar funções criptográficas (incluindo assinatura eletrônica da leitura de instrumentos de medição), garantir a integridade do software durante o controle metrológico legal, ter meios de proteção física do chip contra violação, realizar carga remota de software entre outras funções.

O desenvolvimento de um dispositivo semicondutor, com as especificações citadas acima, apresenta diversos desafios tecnológicos a serem superados. Dentre eles, podemos citar principalmente:

- Facilidade de utilização – a solução apresentada deve possuir baixa dificuldade para utilização dos usuários finais, incentivando o uso e facilitando a validação do software legalmente relevante;
- Baixo custo de implementação – o custo de integração no semicondutor nos medidores atualmente disponíveis deve ser o menor possível, diminuindo assim o impacto para os fabricantes de medidores, incentivando a utilização do mesmo.
- Alta compatibilidade – o dispositivo deve possuir alta compatibilidade com as arquiteturas utilizadas atualmente pelo mercado de medidores.

Uma outra característica fundamental do chip metrológico é possuir mecanismo anti-violação de modo a impedir que terceiros tenham acessos áreas de informações secretas ou sensíveis, como por exemplo as chaves criptográficas ou as constantes de calibração. O mecanismo anti-violação deve, então, ser resistente a todo tipo de ataque físico ao dispositivo.

Na figura 6 mostramos a arquitetura proposta para o chip metrológico, com as seguintes funções:

- Validação da imagem de boot do processador principal;
- Executar, de forma segura, funções criptográficas e em específico realizar a guarda segura de chaves;
- Assinatura dos dados de entrada providos pelo conversor Analógico/Digital;
- Validação dos dados processados pelo processador principal;

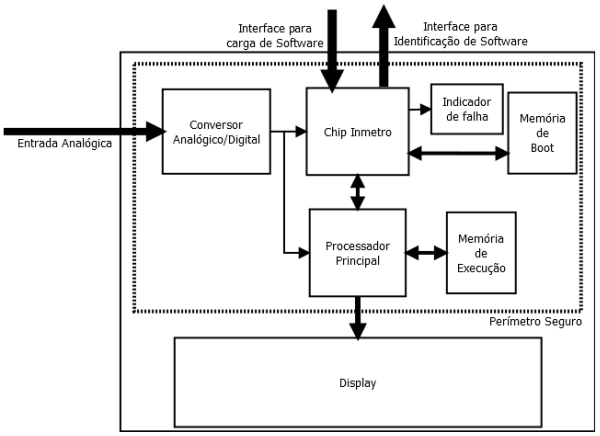


Figura 6. Arquitetura para o chip metrológico

Esta linha de pesquisa resultou num relatório técnico descrevendo a viabilidade técnica e econômica para o desenvolvimento do chip metrológico. Para validação deste estudo, foi desenvolvido um protótipo em FPGA simulando as funcionalidades do chip projetado.

Os resultados obtidos serviram de base para a submissão de uma proposta de P&D, na chamada pública da Eletrobras de 2014, para o desenvolvimento de um cabeça-de-série do chip metrológico.

III. Resultados Gerais obtidos no projeto

Além dos resultados apresentados nas subseções anteriores que são relativas a cada atividade de pesquisa desenvolvida neste projeto de P&D, cabe destacar outros resultados alcançados ao longo deste projeto e abaixo listados:

- Qualificação de Pessoal – este projeto contribui na formação de 2 doutores e 1 mestre.
- Artigos nacionais e internacionais em congressos – ao

longo do projeto foram publicados mais de 10 artigos em congressos nacionais/internacionais;

- Artigo em revista - em termos de publicação, um dos resultados mais relevantes foi a divulgação deste projeto de P&D na revista NCSL International Mesaure [23].
- Gestão de processos – frente aos desafios que cada linha de pesquisa trouxe, tornou-se fundamental criar mecanismos de sistematização da análise de software de modo que as avaliações seguissem um padrão formal. Embora seja um resultado indireto de projeto de P&D, esses mecanismos já vem sendo implementado nos processos internos do Inmetro.

IV. CONCLUSÕES

Com o avanço da informática e da microeletrônica, observa-se, cada vez mais, a presença de dispositivos inteligentes em vários áreas. No campo da metrologia, os equipamentos de medição são, atualmente, baseados em microcontroladores com software embarcado. Neste artigo, mostramos que a validação destes medidores inteligentes será eficaz se o regulador investir no conhecimento científico, no desenvolvimento de ferramentas especializadas e na formação qualificada de recursos humanos (mestres e doutores). Este artigo abordou as seguintes linhas de pesquisa: Análise de Software, Proteção de Software, Cadeias de Confiança de medidores e Chip Metrológico. Cada linha de pesquisa revelou desafios complexos, mas com resultados promissores e que já, atualmente, vem sendo empregados nos processos de avaliação dos medidores inteligentes. Em específico, o estudo de viabilidade do Chip Metrológico aponta para um avanço significativo no processo de avaliação destes medidores inteligentes.

V. REFERÊNCIAS BIBLIOGRÁFICAS

[1] J. C. Mateus and P. E. C. Franco, "Transmission Loss Allocation through Equivalent Bilateral Exchanges and Economical Analysis," *IEEE/PES Powertech*, Saint Petesburg, 2005.

[2] J. C. Mateus and P.E. Cuervo, "Transmission Cost and Loss Allocation Method through Linearized Distribution Factors," *International Conference European Electricity Market (EEM06)*, Varsóvia, p. 413-420, 2006.

[3] NIST - Issues Expanded Draft of Smart Grid Cyber Security Strategy For Public Review and Comment, http://www.nist.gov/smartgrid/smartgrid_020310.cfm (Last accessed, Jan 2014).

[4] C. Collberg and J. Nagra, Surreptitious Software: Obfuscation, Watermarking, and Tamperproofing for Software Protection, Addison Wesley, 2010.

[5] D. R. Boccardo, R. C. S. Machado, L. F. R. Carmo., "Transformações de código para proteção de software", X Simpósio Brasileiro em Segurança da Informação e de Sistemas Computacionais, Minicursos, cap.3, pp. 103-148, Out. 2010.

[6] C. Wang, J. Hill, J. Knight and J. Davidson, (2000) "Software tamper resistance: Obstructing static analysis of programs", Technical Report CS-2000-12, Dept. of Computer Science, University of Virginia.

[7] W. Cho, I. Lee, S. Park (2001) "Against intelligent tampering: Software tamper resistance by extended control flow obfuscation", World Multiconference on Systems, Cybernetics, and Informatics, International Institute of Informatics and Systematics.

[8] T. Ogiso, Y. Sakabe, M. Soshi, and A. Miyaji (2003) "Software obfuscation on a theoretical basis and its implementation", IEEE Trans. Fundamentals, E86-A(1).

[9] A. Lakhotia, D. Boccardo, A. Singh, A. Manacero. (2010) "Context sensitive analysis without calling-context", Journal of Higher-Order and Symbolic Computation, No. 3, V. 23, p. 275-313.

[10] J. Viega, J. and M. Messier (2003) "Secure Programming Cookbook for C and C++", O'Reilly Media publisher.

[11] M. G. de Souza, J. C. T. Baptista, Luci Pirmez, D. R. Boccardo and R. O. Costa (2013) "API para Transformação de Código Assembly", XXXV Jornada Giulio Massarani de Iniciação Científica, Tecnológica, Artística e Cultural UFRJ.

[12] R. O. Costa, D. R. Boccardo, C. Gomes, L. F. R. C. Carmo and L. Pirmez, (2013) "Sensitive Information Protection for Advanced Metering Infrastructure", International Congress on Electrical Metrology – SEMETRO.

[13] L. M. S. Bento, D. R. Boccardo, R. C. S. Machado, V. G. P. Sá, and J. L. Szwarcfiter. Proteção de software por marcas d'água baseadas em grafos. In Anais do XL Seminário Integrado de Software e Hardware (SEMISH'13), volume (forthcoming), 2013.

[14] M. Chroni and S. D. Nikolopoulos. Encoding watermark integers as self-inverting permutations. In Proceedings of the 11th International Conference on Computer Systems and Technologies and Workshop for PhD Students in Computing on International Conference on Computer Systems and Technologies, CompSysTech'10, pages 125–130, New York, NY, USA, 2010. ACM.

[15] M. Chroni and S. D. Nikolopoulos. An e_cient graph codec system for software watermarking. 36th IEEE Conference on Computers, Software, and Applications (COMPSAC'12), 0:595–600, 2012.

[16] L. M. S. Bento, D. R. Boccardo, R. C. S. Machado, V. G. P. Sá, and J. L. Szwarcfiter. Towards a provably resilient scheme for graph-based watermarking. In Andreas Brandstadt, Klaus Jansen, and Rudiger Reischuk, editors, WG, volume 8165 of Lecture Notes in Computer Science, pages 50–63. Springer, 2013.

[17] L. M. S. Bento, D. R. Boccardo, R. C. S. Machado, V. G. P. Sá, and J. L. Szwarcfiter. Fingerprinting de Software e Aplicações à Metrologia Legal. In: International Congress on Electrical Metrology - SEMETRO, 2013, Buenos Aires. International Congress on Electrical Metrology - SEMETRO, 2013

[18] E. W. Dijkstra, Notes on Structured Programming, in Structured Programming (1972), 1-82, Acad. Press.

[19] L. M. S. Bento, D. R. Boccardo, R. C. S. Machado, V. G. P. Sá, and J. L. Szwarcfiter. The Graphs of Structured Programming. Submetido - 13th Cologne-Twente Workshop on Graphs & Combinatorial Optimization, 2015.

[20] L. M. S. Bento, D. R. Boccardo, R. C. S. Machado, V. G. P. Sá, and J. L. Szwarcfiter A randomized graph-based scheme for software watermarking. In Anais do XIV Simpósio Brasileiro de Segurança da Informação e de Sistemas Computacionais (SBSEG'14), Belo Horizonte (MG), nov 2014.

[21] V. G. P. de Sá, D. R. Boccardo, L. F. Rust and R. C. S. Machado, "A tight bound for exhaustive key search attacks against Message Authentication Codes," *RAIRO - Theoretical Informatics and Applications*, 2013.

[22] F. Douglas, "The compression cache: using on-line compression to extend physical memory," *in Proc. 3rd USENIX Conference*, pp. 519–529, 1993.

[23] S. Camara, R. Machado e L. Carmo, "A Consumption Authenticator Based Mechanism for Time-of-Use Smart Meter Measurements Verification,", Appl. Mech. Mater., vol. 241-244, pp. 218-22, 2013.

[24] C. B. Prado. ; D. R. Boccardo.; R. C. S. Machado; L. F. R. C. Carmo; T. M. Nascimento; L. M. S. Bento; R. O. Costa,; C. G. Castro; S. M. Camara,. Smart Metering CyberSecurity. NCSL International Measure: the journal of measurement science, 2014.