Análise de criptografia no contexto da avaliação da conformidade*

Lucila Maria de Souza Bento[1], **Raphael Carlos Santos Machado**[1,2,3]

[1] Instituto Nacional de Metrologia, Qualidade e Tecnologia (INMETRO)
Duque de Caxias, RJ – Brasil

[2]Programa de Pós-Graduação em Ciência da Computação
Centro Federal de Educação Tecnológica Celso Suckow da Fonseca
Rio de Janeiro, RJ – Brasil

[3]Instituto de Computação
Universidade Federal Fluminense (UFF)
Niterói, RJ – Brasil

{lmbento, rcmachado}@inmetro.gov.br

Abstract. *To protect sensitive data processed by embedded systems, developers have to rely on cryptography. While cryptographic mechanisms have become increasingly advanced, many data breaches occur because developers do not correctly use the cryptography. This paper present a list (not exhaustive) of incorrect use of cryptography in embedded systems to complement the analyzes performed in the conformity assessments of these systems.*

Resumo. *Para proteger informações sensíveis processadas por sistemas embarcados, os desenvolvedores precisam confiar na criptografia. Embora os mecanismos criptográficos estejam cada vez mais avançados, muitas violações de segurança ocorrem porque os desenvolvedores usam a criptografia incorretamente. Este artigo apresenta uma lista (não exaustiva) de uso incorreto de criptografia em sistemas embarcados para incrementar as análises realizadas num processo de avaliação da conformidade destes sistemas.*

1. Introdução

O crescimento no uso de dispositivos inteligentes está tornando cada vez mais reconhecido o fato de que a segurança da informação deve ocupar um papel central no projeto de sistemas de tecnologia da informação (TI). Atualmente, um grande volume de dados, dos mais diversos tipos, é armazenado, processado e transmitido por aplicações de TI executadas como sistemas embarcados. Algumas dessas aplicações dependem fortemente de mecanismos de segurança por estarem inseridas em relações de consumo e, consequentemente, sujeitas a situações adversas como tentativas de fraude, por exemplo, e/ou utilizarem canais de comunicação inseguros, como redes sem fio, por exemplo. Alguns exemplos desses sistemas embarcados incluem medidores de energia elétrica, aparelhos hospitalares, TV por assinatura, bombas de combustível, aparelhos celulares e veículos.

Dentre os mecanismos de segurança da informação adotados em projetos de sistemas embarcados, a criptografia é protagonista na solução de muitos problemas [NIST 2002]. No Brasil, inclusive, existem documentos regulatórios nos

*Trabalho apoiado por CNPq, Faperj e Projeto SHCDCiber.

quais são descritos requisitos de segurança que prevê o uso de criptografia para prover serviços de segurança (integridade, autenticidade, confidencialidade e não-repúdio [Aumasson 2017]) como é o caso do Regulamento Técnico Metrológico (RTM) para medidores de energia elétrica, por exemplo. Os sistemas embarcados cobertos por tais documentos regulatórios precisam passar por um processo de avaliação da conformidade a fim de verificar se estão de acordo com os requisitos estabelecidos. Assim, é fundamental que os responsáveis pela realização do processo de avaliação sejam capazes de avaliar como os mecanismos criptográficos foram utilizados e se os serviços de segurança pretendidos com seu uso foram obtidos apropriadamente.

A implementação de mecanismos criptográficos apresenta vários requisitos e desafios. O desempenho dos algoritmos é frequentemente crucial, devido as limitações de hardware e taxas de transmissão dos links de comunicação. Além disso, a garantia de segurança é outro grande desafio, que muitas vezes varia entre a flexibilidade e o custo reduzido da execução dos mecanismos criptográficos em um computador de uso geral, e o uso de hardwares criptográficos encapsulados com segurança para evitar que invasores violem o sistema. Desta forma, possivelmente pela complexidade dos conceitos associados a criptografia e o esforço necessário à sua implementação, observamos muitas pesquisas voltadas a análise de características dos mecanismos criptográficos e quais são mais apropriados para uso em sistemas embarcados, mas poucos esforços no sentido de avaliar a corretude da implementação destes mecanismos, sendo os esforços existentes concentrados, essencialmente, no uso de ferramentas automatizadas.

Neste contexto, o presente artigo tem como objetivo contribuir para a melhoria do processo de avaliação da conformidade de mecanismos criptográficos, por meio da apresentação de uma lista não exaustiva de erros e fragilidades no uso de criptografia introduzidos durante o projeto e a implementação de sistemas embarcados. A ideia é que a lista apresentada apoie atividades de revisão manual de código-fonte com foco na análise do uso de criptografia para atendimento a requisitos de Segurança da Informação previstos em documentos regulatórios. Além disso, são propostas medidas passíveis de serem adotadas para identificação e mitigação dos erros e fragilidades listados.

2. Trabalhos relacionados

Dentre os estudos que avaliam vulnerabilidades associadas ao uso de criptografia, o trabalho de Lazar et al. [Lazar et al. 2014] merece destaque, pois os autores mostraram que somente 17% das vulnerabilidades de criptografia estão em bibliotecas de software e os outros 83% correspondem ao uso incorreto das bibliotecas.

Nesse seguimento, Shuai et al [Shuai et al. 2014] realizaram uma análise sistemática sobre o uso incorreto de criptografia em aplicativos Android e construíram um protótipo de ferramenta para identificação de tais erros de uso. Arzt et al. [Arzt et al. 2015] apresentaram uma solução que identifica automaticamente os algoritmos criptográficos corretos a serem usados, com base nas respostas do desenvolvedor a perguntas de alto nível, e retorna um código Java correspondente e um protocolo de uso desse código que descreve as restrições de uso da API.

Braga e Dahab [Braga and Dahab 2015] abordam o uso de ferramentas e técnicas de segurança durante os vários estágios de desenvolvimento de mecanismos criptográficos. E em [Braga et al. 2017], os autores propõe um método de *benchmarking* de fer-

ramentas de análise estática de código para a detecção de uso incorreto de criptografia e avalia o método proposto por meio de um estudo de caso no qual foi identificado que todas as ferramentas avaliadas juntas detectaram apenas 35% de uso incorreto de criptografia. Esse resultado indica que são necessários esforços adicionais para melhorias nos métodos usados para detecção do uso incorreto de criptografia.

Adicionalmente, temos que os padrões atuais de codificação segura, considerando as listagens de vulnerabilidades mais frequentes [OWASP 2018, MITRE 2019], fornecem orientações/regras simples e gerais para evitar erros e/ou fragilidades no uso de criptografia. Tais orientações/regras estão implementadas em ferramentas de análise estática disponíveis no mercado, como SonarQube[1], FindSecBugs[2] e Yasca[3], por exemplo.

3. Análise do uso incorreto de criptografia

Conforme visto na Seção 2, a avaliação de erros e/ou fragilidades no uso de criptografia ainda corresponde a um grande desafio. Assim, a presente seção apresenta um conjunto de fragilidades e erros comuns encontrados no projeto e implementação de mecanismos criptográficos em sistemas embarcados, que pode ser utilizado para complementar as análises efetuadas em um processo de avaliação da conformidade, incluindo aqueles que adotam ferramentas automatizadas disponíveis no mercado.

Note que a lista está dividida por assunto e não tem o propósito de ser exaustiva, sendo apresentada, unicamente, com base no entendimento de relevância dos autores. Sempre que apropriado, será informado o código CWE da violação em questão. Observe ainda que, em grande parte, as fragilidades apresentadas a seguir estão relacionadas à vulnerabilidade "*A3-Sensitive Data Exposure*" do OWASP TOP10-2017[4].

3.1. Chaves Criptográficas

As chaves criptográficas tem papel fundamental na garantia da segurança das informações protegidas pelo uso de criptografia, na efetividade dos protocolos e mecanismos de segurança, e na proteção de outras chaves. Todas as chaves devem ser protegidas contra modificação, e chaves secretas devem ser protegidas contra divulgação não-autorizada.

O gerenciamento de chaves deve prover os fundamentos para geração, armazenamento, distribuição e destruição de chaves. A publicação NIST SP 800-57 [Barker 2016] orienta sobre o gerenciamento de chaves criptográficas. Informações adicionais são fornecidas no NIST FIPS 140 [NIST 2002], o qual apresenta requisitos mínimos para módulos criptográficos que incorporam funções de gerenciamento de chaves. No entanto, é comum encontrar problemas associados ao gerenciamento de chaves, e algumas fragilidades e erros comuns são mostradas a seguir.

1. **Uso de chaves criptográficas "*hard coded*".** Esse erro consiste em armazenar chaves criptográficas, ou *hash* de chaves criptográficas, em texto claro no código-fonte da aplicação ou em arquivo de configuração desprotegido, que podem ser recuperados por agentes maliciosos, por meio de engenharia reversa, por exemplo.

[1] `https://www.sonarqube.org/`
[2] `https://find-sec-bugs.github.io/`
[3] `https://www.scovetta.com/yasca/`
[4] `https://owasp.org/index.php/Top_10-2017_A3-Sensitive_Data_Exposure`

O problema também ocorre com senhas e informações sensíveis em geral (CWE-321[5]), e pode ser identificado com o uso de ferramentas de análise estática capazes de detectar as ocorrências dessa fragilidade, como o SonarQube, por exemplo, ou por meio da busca por strings codificadas na aplicação ou armazenadas em arquivo de configuração desprotegido. Para mitigação as chaves criptográficas (ou senhas) em arquivos de configuração podem ser armazenados em arquivos de configuração devidamente protegidos, ou em sistema de gerenciamento de chaves seguro capaz de controlar a sequência de estados em seu ciclo de vida e permitir que somente um administrador autorizado possa manipular as chaves.

2. **Uso de chaves criptográficas "pequenas".** Toda chave criptográfica deve ser, direta ou indiretamente, saída de um gerador de números aleatórios aprovado e devem ser longas o suficiente para resistir a ataques de força bruta. A identificação do uso de chaves criptográficas "pequenas" pode ser realizada por meio da análise da documentação do produto de software e, complementada com a inspeção manual do código fonte, identificando e examinando os pontos onde as chaves criptográficas são definidas. A mitigação da fragilidade pode ser obtida por meio de orientações disponíveis em guias de recomendação internacional para comprimento de chave [BSI 2019, Barker and Roginsky 2019].
3. **Uso de chaves criptográficas expiradas.** O tempo de vida de uma chave criptográfica é proporcional a sua "força". Portanto, utilizar chaves criptográficas expiradas pode comprometer a segurança dos dados criptografados. Para identificar essa fragilidade é necessário analisar a documentação do produto de software e inspecionar o código-fonte em busca de informações sobre o vencimento das chaves. O ideal é que a aplicação possua um mecanismos de substituição das chaves criptográficas ligeiramente antes de sua expiração, ou utilize um gerenciador de chaves seguro que realize as ações apropriadas para troca de chaves em operação.
4. **Uso de senhas como chaves criptográficas.** Devido à baixa entropia e à possivelmente pobre aleatoriedade de tais senhas, elas não são apropriadas para serem utilizadas diretamente como chaves. O uso de senhas como chaves criptográficas pode ser identificado por meio da análise da documentação e do código-fonte, avaliando os pontos em que as chaves são definidas, ou com o uso de ferramentas de avaliação de criptografia, como o Cryptosense[6], por exemplo. Para mitigação dessa fragilidade, pode-se usar chaves geradas de acordo com as recomendações de especialistas [BSI 2019, Barker and Roginsky 2019]. Caso seja desejado derivar chaves a partir de uma senha ou frase-passe, recomenda-se o uso do documento NIST SP 800-132 [Turan et al. 2010] como referência.
5. **Chaves não limpas adequadamente após o uso.** É necessário ter um cuidado especial com as variáveis que armazenam/manipulam chaves criptográficas, em particular com o tempo em que essas variáveis persistem no código e como são limpas após o uso, de modo que as chaves sejam mantidas em memória somente durante o período imprescindível para seu uso e, após esse período, devem ser removidas adequadamente da memória. A detecção desse problema pode ser realizado por meio da análise do código-fonte, percorrendo todo o fluxo das chaves utilizadas a fim de garantir que estejam sendo mantidas em memória somente o

[5] https://cwe.mitre.org/data/definitions/321.html
[6] https://cryptosense.com/

tempo necessário e se podem ser, de alguma forma, recuperadas após remoção.

6. **Gerenciamento inadequado de chaves.** Os erros no gerenciamento de chaves são comuns e ocorrem quando existe falhas no seu projeto ou implementação, são adotados gerenciadores de chaves fracos ou mal configurados, assim como problemas em módulos de segurança de hardware que, de alguma forma, estejam relacionados com o ciclo de vida das chaves criptográficas. Geralmente, os desenvolvedores assumem que o armazenamento de chaves não será examinado por um usuário arbitrário, mas muitos usuários de uma aplicação têm acesso ao registro, arquivos temporários e bancos de dados, sendo possível que esses usuários acessem as chaves usando arquivos temporários, ocultos e de registro. A detecção desse problema é um processo trabalhoso e demanda a análise do escopo da aplicação em questão, exame dos processos de negócio internos e revisão das medidas adotadas para garantir que as práticas recomendadas de gerenciamento de chaves, como [Barker et al. 2013, DCSSI 2006], estejam sendo seguidas.
7. **Troca de chaves insegura.** O protocolo de troca de chaves descreve a maneira usada para trocar chaves ou outras informações necessárias para estabelecer um canal de comunicação seguro. Acontece que, principalmente quando adotado um algoritmo simétrico, se as partes não puderem estabelecer uma troca de chaves inicial segura, as mensagens trafegadas entre as partes podem ser interceptadas e decriptadas por terceiros que adquiriram a chave durante a troca de chave inicial. A revisão da documentação e do código-fonte do produto de software é fundamental para identificar se a troca de chaves está sendo realizada de maneira segura. Uma boa alternativa para implementar uma troca de chaves segura é utilizar algum protocolo sugerido por especialistas [Barker et al. 2018, Barker et al. 2019], uma vez que estes foram testados e a segurança avaliada.

3.2. Geração de Números Aleatórios

Geradores de números aleatórios (RNGs) são fundamentais em protocolos de segurança. Eles são usados, por exemplo, para a geração de chaves e vetores de inicialização (IV), para a identificação única de mensagens, entre outros. Existem duas classes de RNGs, a saber, determinísticos e não-determinísticos. Geradores determinísticos, conhecidos como *geradores de números pseudo-aleatórios* (PRNG, do inglês *pseudo-random number generator*), usam algoritmos criptográficos para gerar uma sequência de bits "aparentemente" aleatórios — dos quais podem ser obtidos números aleatórios — a partir de uma semente. Neste sentido, o número ou sequência de bits não é, de fato, aleatória — apenas aparenta ser, aos olhos de quem desconhece a semente. Adicionalmente, por usar um parâmetro para a geração da sequência de bits, os PRNG poderiam ser classificados — e alguns autores o fazem — como algoritmos de chave simétrica (ou chave única). Geradores não-determinísticos, conhecidos como *geradores de números aleatórios verdadeiros* (TRNG, do inglês *true random number generator*), produzem saída a partir de fontes físicas imprevisíveis, fora do controle humano, como, por exemplo, o decaimento radioativo ou o ruído térmico de um equipamento[7].

1. **Uso de algoritmo gerador de números pseudo-aleatórios "não-criptográfico" com finalidades de segurança.** Quando um gerador de números pseudo-

[7]Cabe observar que, na verdade, o ruído térmico de um equipamento pode não ser completamente imprevisível, uma vez que poderá depender, por exemplo, do uso que se faz deste equipamento.

aleatórios não é projetado para criptografia e é usado em um contexto criptográfico, ele pode expor a criptografia a certos tipos de ataques, em particular aos ataques de força bruta. Alguns exemplos de geradores fracos consistem no tempo do relógio do sistema, uso de um contador em vez de um número aleatório, acesso ao gerador de números aleatórios antes que o sistema tenha acumulado entropia suficiente ou o uso de parâmetros em *hard coded* utilizados durante testes. Para identificar esse problema deve-se revisar a documentação e o código-fonte em busca de funções de geração de número aleatórios disponíveis na linguagem de programação adotada ou para análise de qualquer função que objetiva a geração de números aleatórios e/ou funções criptográficas. Uma possível medida para mitigação da fragilidade é a adoção de funções ou hardware que usam uma geração de números aleatórios baseada em hardware. Por exemplo, `CyptGenRandom` no Windows ou `hw_rand()` no Linux (CWE-338[8]).

2. **Uso de fontes de entropia fraca.** O uso de sementes previsíveis reduz significativamente o esforço de um agente malicioso para prever quais números aleatórios serão gerados por um gerador de números pseudo-aleatório dado (CWE-337[9]). Alguns exemplos incluem o uso de registrador de horário (como `System.currentTimeMillis()` em Java e `time()` em C) ou o número de identificação de processos. Portanto, essa fragilidade pode ser descoberta ao avaliar todas as funções que objetivam a geração de números aleatórios e/ou funções criptográficas, tanto na documentação do produto de software quanto no código-fonte. Convêm buscar também por funções da linguagem de programação adotada que retornam o horário do sistema, pois correspondem a uma semente incorreta bastante utilizada. Mas o problema pode ser evitado com o uso de entradas não previsíveis para geração de sementes. Pode ser considerado o uso de produtos ou módulos em conformidade com o FIPS 140-2 [NIST 2002], e/ou use um gerador de números pseudo-aleatórios recomendados por especialistas [NIST 2000, Institute and Association 1998, NIST 2000, Bassham III et al. 2010].

3.3. Resumos Criptográficos (funções *hash*)

Um resumo criptográfico é uma representação curta de uma mensagem. Uma função hash é uma transformação para a qual é fácil computar o resumo, mas é difícil reverter o processo, ou seja, encontrar uma mensagem com um dado resumo. Adicionalmente, espera-se que seja difícil encontrar quaisquer duas mensagens com mesmo resumo criptográfico. Assim, uma boa função hash é dotada das seguintes propriedades:

Resistência a ataques de pré-imagem. Dado um resumo criptográfico d escolhido aleatoriamente, é computacionalmente inviável encontrar uma mensagem x com resumo criptográfico $h(x) = d$, onde h é a função hash.

Resistência a colisões. É computacionalmente inviável encontrar duas entradas distintas x e x' para h que originem o mesmo resumo criptográfico $h(x) = h(x')$.

Resistência de segunda ordem a ataques de pré-imagem. Dada uma mensagem x escolhida aleatoriamente, é computacionalmente inviável encontrar uma segunda mensagem y com mesmo resumo criptográfico $h(x) = h(y)$.

[8] `https://cwe.mitre.org/data/definitions/338.html`
[9] `https://cwe.mitre.org/data/definitions/337.html`

Um resumo ou *hash não-criptográfico* é uma função que mapeia sequências de comprimento arbitrário em sequências de tamanho fixo, mas sem os requisitos de resistência a colisão de um hash criptográfico. A propriedade mais importante de um algoritmo de dispersão (hash não-criptográfico) é o fato de uma sequência escolhida aleatória e uniformemente ser mapeada (aproximadamente) uniformemente no espaço dos resumos. Esta propriedade torna tais algoritmos úteis em aplicações de estruturas de dados. Portanto, a fragilidade a seguir é comumente encontrada.

1. **Uso de algoritmo de hash "não-criptográfico" com finalidades de segurança.** Conforme visto, algoritmos de hash "não criptográficos" fornecem garantias mais fracas de colisão em troca de melhorias de desempenho e não possuem o mesmo rigor que os algoritmos de hash criptográficos, permitindo recuperar uma mensagem dado seu resumo com maior facilidade. O uso de tais algoritmos pode ser detectado por meio da revisão da documentação e do código-fonte do produto de software em questão, ou com o uso de ferramentas de análise de criptografia, como a ferramenta Cryptosense, por exemplo. Sendo recomendado, no entanto, o emprego de hash criptográficos recomendados por especialistas [NIST 2015].

3.4. Outros erros comuns no uso de mecanismos criptográficos

A seguir são abordados problemas associados ao uso de algoritmos criptográficos cuja segurança é desconhecida ou sabidamente frágil, e questões relacionadas a falta de conhecimento abrangente dos mecanismos criptográficos adotados.

1. **Uso de algoritmos criptográficos próprios.** O uso de algoritmos criptográficos próprios é um risco desnecessário que pode resultar na exposição de informações sensíveis, pois eles provavelmente estão expostos a ataques bem compreendidos por criptógrafos. A melhor alternativa, portanto, é sempre utilizar algoritmos criptográficos padrão recomendados por especialistas [BSI 2019, NIST 2002].
2. **Uso de algoritmos criptográficos fracos.** Com o passar do tempo e os avanços na criptoanálise e na capacidade de computação, os algoritmos criptográficos se tornam obsoletos. Com isso, tão perigoso quanto utilizar algoritmos criptográficos próprios é utilizar algoritmos criptográficos antigos, podendo ser ou não em bibliotecas antigas ou componentes de *framework* (CWE-326[10] e CWE-327[11]). O uso de tais algoritmos, chamados de algoritmos não padrão (exemplo ES, DES, MD5, SHA-1, entre outros), é perigoso porque um agente malicioso pode conhecer técnicas capazes de quebrar o algoritmo e comprometer quaisquer dados que tenham sido protegidos. Por isso, é recomendado usar algoritmos bem controlados que sejam considerados atualmente fortes por especialistas da área e implementações bem testadas [BSI 2019, NIST 2002], além de ser necessário assegure-se periodicamente que os mecanismos criptográficos utilizados não tornaram-se obsoletos e projetar o produto de software para que um algoritmo criptográfico possa ser substituído por outro, a fim de facilitar a atualização para algoritmos mais fortes.
3. **Escolha incorreta de parâmetros para algoritmos criptográficos.** Alguns algoritmos criptográficos utilizam *parâmetros de domínio*, que são valores adicionais necessários à adequada operação do algoritmo criptográfico. Deste modo,

[10] `https://cwe.mitre.org/data/definitions/326.html`
[11] `https://cwe.mitre.org/data/definitions/327.html`

algoritmos criptográficos são seguros apenas quando são usados com os parâmetros corretos, tornando a definição destes parâmetros uma tarefa crucial para a garantia da segurança. Descobrir essa fragilidade pode ser difícil, principalmente quando são adotadas bibliotecas de criptografia que possuem valores padrão ocultos para os parâmetros não especificados, sendo necessário, portanto, revisar detalhadamente os algoritmos criptográficos adotados, sejam eles implementados pelo próprio fabricante ou disponibilizados em bibliotecas ou *frameworks*, inspecionando os parâmetros necessários ao correto funcionamento de tais algoritmos e considerando todas as características recomendadas por especialistas [Barker 2016, Barker 2006]. Também pode-se considerar a utilização de algoritmos criptográficos disponíveis em bibliotecas amplamente testadas e validadas, cujos parâmetros estejam claramente definidos.

4. **Uso incorreto/inadequado de modos de encriptação.** Existem diferentes modos de encriptação disponíveis, isto é, diversas maneiras de usar uma cifra de blocos, como o AES, por exemplo, e cada uma delas pode ser adotada em cenários específicos. Um erro comum consiste em adotar um modo de encriptação em um cenário não favorável às suas características de funcionamento. Por exemplo, sabe-se que com o ECB (*Electronic Code Book*) os blocos de mensagem idênticos vão produzir blocos cifrados idênticos, portanto seria um erro adota-lo, por exemplo, num cenário em que existem muitos blocos de mensagem idênticos e pretende-se distinguir blocos diferentes apesar de seus conteúdos serem iguais. Logo, identificar esses casos exige a avaliação do escopo da aplicação em questão, os processos de negócio internos e como os modos de encriptação foram concebidos/implementados, a fim de identificar possíveis cenários em que um dado modo de encriptação foi usado e suas características não sejam suficientes ou interessantes para atingir os objetivos buscados com tal implementação. Adicionalmente, é recomendado utilizar somente os modos de operação de cifra de blocos recomendados e descritos na literatura [Dworkin 2010a, Dworkin 2005, Dworkin 2007a, Dworkin 2007b, Dworkin 2010b, Dworkin 2012].
5. **Uso de criptografia sem propósito claro ou bem-definido.** Apesar da criptografia ser uma ferramenta extremamente útil para segurança em diversos aspectos, não é difícil encontrar desenvolvedores misturando diversos mecanismos criptográficos sem um objetivo claro. Para utilizar corretamente a criptografia, sua utilidade deve surgir de maneira natural, para atender a uma necessidade clara e bem definida, caso contrário, corre-se o risco de desprender grande esforço para implementação de mecanismos criptográficos sem que isso seja, de fato, necessário ou que, sequer, contribua para a segurança do produto de software. Uma avaliação do projeto, revisão do escopo da aplicação, dos processos de negócio, das necessidades de comunicação do produto de software, da documentação e código-fonte da aplicação são imprescindíveis para certificar-se de que os mecanismos criptográficos adotados possuem um papel/objetivo claro.
6. **Uso incorreto de criptografia.** É necessário entender bem cada um dos modos de operação de criptografia para utilizar os mecanismos criptográficos de maneira adequada. Por exemplo, um erro comum é assumir que o uso de criptografia fornecerá integridade à mensagem. Note que a criptografia oculta dados, mas um invasor pode modificar os dados criptografados e os resultados podem ser aceitos pelo produto de software se a integridade da mensagem não for verificada.

Tal fato pode causar um comportamento adverso na aplicação e um agente malicioso pode valer-se desse cenário para efetuar um ataque real à aplicação. Em outras palavras, implementar um mecanismo criptográfico de maneira correta e usá-lo de maneira errada resultará em um sistema que é facilmente comprometido. Em vista disso, assim como no Item 5, para identificar se a criptografia está sendo usada corretamente é necessário realizar uma análise completa do escopo da aplicação, processos de negócio, necessidades de comunicação do produto de software, documentação apresentada e código-fonte, a fim de compreender se os mecanismos criptográficos considerados e implementados realmente solucionam os problemas que o fabricante pretende resolver com os mesmos.

4. Conclusões e trabalhos futuros

Os erros e fragilidades no uso de criptografia listados podem ser observados não só em sistemas embarcados, de modo que as informações e recomendações apresentadas para identificação e mitigação dos problemas apontados persistem válidas em outros cenários.

O texto apresenta resultados preliminares de um esforço para construção de um guia que possa ser usado para avaliação de mecanismos criptográficos implementados em sistemas embarcados cobertos por algum documento regulatório e, consequentemente, que devem passar por um processo de avaliação da conformidade para garantir o atendimento aos requisitos do documento. Com isso, pretende-se expandir a lista de erros e fragilidades apresentada, e realizar estudos de casos para avaliar a eficiência das análises conduzidas por meio das verificações provenientes desta lista na identificação de vulnerabilidades associadas ao uso incorreto de criptografia.

Referências

Arzt, S., Nadi, S., Ali, K., Bodden, E., Erdweg, S., and Mezini, M. (2015). Towards secure integration of cryptographic software. In *2015 ACM International Symposium on New Ideas, New Paradigms, and Reflections on Programming and Software*, Onward! 2015, pages 1–13, New York, NY, USA. ACM.

Aumasson, J.-P. (2017). *Serious Cryptography: A Practical Introduction to Modern Encryption*. No Starch Press, San Francisco, CA, USA.

Barker, E., Chen, L., Roginsky, A., Vassilev, A., and Davis, R. (2018). Sp 800-56a. recommendation for pair-wise key-establishment schemes using discrete logarithm cryptography - revision 3. Technical report, NIST, Gaithersburg, MD, United States.

Barker, E., Chen, L., Roginsky, A., Vassilev, A., Davis, R., and Simon, S. (2019). Sp 800-56b. recommendation for pair-wise key-establishment schemes using integer factorization cryptography - revision 2. Technical report, NIST, Gaithersburg, MD, United States.

Barker, E., Smid, M., and Chokhani, B. S. (2013). Nist special publication 800-131a. a framework for designing cryptographic key management systems. Technical report, NIST, Gaithersburg, MD, United States.

Barker, E. B. (2006). Sp 800-89. recommendation for obtaining assurances for digital signature applications. Technical report, NIST, Gaithersburg, MD, United States.

Barker, E. B. (2016). Sp 800-57. recommendation for key management, part 1: General revision 4. Technical report, NIST, Gaithersburg, MD, United States.

Barker, E. B. and Roginsky, A. L. (2019). Draft nist special publication 800-131a revision 2, [forthcoming]. transitioning the use of cryptographic algorithms and key lengths. Technical report, NIST, Gaithersburg, MD, United States.

Bassham III, L. E., Rukhin, A. L., Soto, J., Nechvatal, J. R., Smid, M. E., Barker, E. B., Leigh, S. D., Levenson, M., Vangel, M., Banks, D. L., Heckert, N. A., Dray, J. F., and Vo, S. (2010). Sp 800-22 rev. 1a. a statistical test suite for random and pseudorandom number generators for cryptographic applications. Technical report, NIST, Gaithersburg, MD, United States.

Braga, A. and Dahab, R. (2015). A survey on tools and techniques for the programming and verification of secure cryptographic software. In *Anais do XV Simpósio Brasileiro em Segurança da Informação e de Sistemas Computacionais (SBSeg 2015)*, pages 30–43.

Braga, A., Dahab, R., Antunes, N., Laranjeiro, N., and Vieira, M. (2017). Practical evaluation of static analysis tools for cryptography: Benchmarking method and case study. In *2017 IEEE 28th International Symposium on Software Reliability Engineering (ISSRE)*, pages 170–181.

BSI (2019). Bsi tr-02102-1 - cryptographic mechanisms: Recommendations and key lengths. Technical report, Federal Office for Information Security, Bonn, Germany.

DCSSI (2006). Mécanismes cryptographiques: Règles et recommandations concernant le choix et le dimensionnement des mécanismes cryptographiques de niveau de robustesse standard. Technical report, Laboratoire de Cryptographie de la DCSSI, Paris, France.

Dworkin, M. (2005). Sp 800-38b. recommendation for block cipher modes of operation: The cmac mode for authentication. Technical report, NIST, Gaithersburg, MD, United States.

Dworkin, M. (2007a). Sp 800-38c. recommendation for block cipher modes of operation: The ccm mode for authentication and confidentiality. Technical report, NIST, Gaithersburg, MD, United States.

Dworkin, M. (2007b). Sp 800-38d. recommendation for block cipher modes of operation: Galois/counter mode (gcm) and gmac. Technical report, NIST, Gaithersburg, MD, United States.

Dworkin, M. (2010a). Sp 800-38a. recommendation for block cipher modes of operation: Three variants of ciphertext stealing for cbc mode. Technical report, NIST, Gaithersburg, MD, United States.

Dworkin, M. (2010b). Sp 800-38e. recommendation for block cipher modes of operation: The xts-aes mode for confidentiality on storage devices. Technical report, NIST, Gaithersburg, MD, United States.

Dworkin, M. (2012). Sp 800-38f. recommendation for block cipher modes of operation: Methods for key wrapping. Technical report, NIST, Gaithersburg, MD, United States.

Institute, A. N. S. and Association, A. B. (1998). *ANSI X9.31:1998: Digital Signatures Using Reversible Public Key Cryptography for the Financial Services Industry (rDSA)*. American Bankers Association.

Lazar, D., Chen, H., Wang, X., and Zeldovich, N. (2014). Why does cryptographic software fail?: A case study and open problems. In *Proceedings of 5th Asia-Pacific Workshop on Systems*, APSys '14, pages 7:1–7:7, New York, NY, USA. ACM.

MITRE (2019). Cwe list version 3.3. `https://cwe.mitre.org/`.

NIST (2000). Fips pub 186-2, digital signature standard (dss). U.S.Department of Commerce/National Institute of Standards and Technology.

NIST (2002). Fips pub 140-2, security requirements for cryptographic modules. U.S.Department of Commerce/National Institute of Standards and Technology.

NIST (2015). Fips pub 180-4, secure hash standard (shs). U.S.Department of Commerce/National Institute of Standards and Technology.

OWASP (2018). Open web application security project – top 10-2017 top 10. `https://www.owasp.org/index.php/Top_10-2017_Top_10`.

Shuai, S., Guowei, D., Tao, G., Tianchang, Y., and Chenjie, S. (2014). Modelling analysis and auto-detection of cryptographic misuse in android applications. In *2014 IEEE 12th International Conference on Dependable, Autonomic and Secure Computing*, pages 75–80.

Turan, M. S., Barker, E. B., Burr, W. E., and Chen, L. (2010). Sp 800-132. recommendation for password-based key derivation: Part 1: Storage applications. Technical report, NIST, Gaithersburg, MD, United States.