INSTITUTO FEDERAL DE SANTA CATARINA

PAULO FYLIPPE SELL

Atualização e Carga de *Firmware* em Sistemas Embarcados de Forma Segura e Confiável

São José - SC

outubro/2019

ATUALIZAÇÃO E CARGA DE *FIRMWARE* EM SISTEMAS EMBARCADOS DE FORMA SEGURA E CONFIÁVEL

Trabalho de conclusão de curso apresentado à Coordenadoria do Curso de Engenharia de Telecomunicações do campus São José do Instituto Federal de Santa Catarina para a obtenção do diploma de Engenheiro de Telecomunicações.

Orientador: Emerson Ribeiro de Mello

Coorientador: Roberto de Matos

São José - SC

outubro/2019

RESUMO

Sistemas embarcados estão presentes em grande parte do dia-a-dia das pessoas. Equipamentos que monitoram a saúde, controlam casas e carros inteligentes ou ainda cidades inteiras permitem uma melhor qualidade de vida dos cidadãos. Desta forma, estes dispositivos precisam manter seus *firmwares* atualizados e seguros, evitando que ataques maliciosos que comprometam seu funcionamento sejam realizados. A atualização, por sua vez, deve ser realizada de tal maneira que seja garantido neste processo que apenas *firmwares* de fontes conhecidas sejam recebidos, bem como haja a certeza de que o dispositivo nunca ficará inutilizável após o processo de atualização. A criptografia de chave pública e algoritmos de *hash* podem ser usados para prover tais garantias. À vista do que foi posto, este trabalho irá propor uma solução de atualização de *firmware* em sistemas embarcados microcontrolados, de forma que as propriedades de segurança e confiabilidade sejam garantidas.

Palavras-chave: Atualização de *firmware*. Segurança. Confiabilidade. Integridade.

LISTA DE ILUSTRAÇÕES

Figura 1 – Modelo de criptografia simétrica 15
Figura 2 – Modelo de criptografia assimétrica 15
Figura 3 – Processo de assinatura digital 16
Figura 4 – Arquitetura simplificada de um *Trusted Execution Environment* (TEE) 19
Figura 5 – Componentes de um sistema com *bootloader* 21
Figura 6 – Processo padrão de atualização de *firmware* 21
Figura 7 – Componentes do dispositivo proposto. 23
Figura 8 – Diagrama de atividade da proposta para atualização de *firmware*. 24

LISTA DE TABELAS

Tabela 1 – Campos de um certificado digital - recomendação X.509 16
Tabela 2 – Comparação entre os tipos de criptografia e as propriedades básicas de segurança . . . 17
Tabela 3 – Comparação entre os tipos de criptografia e ataques 17
Tabela 4 – Comparação entre tecnologias de ambientes seguros de execução 20
Tabela 5 – Cronograma de atividades 25

LISTA DE ABREVIATURAS E SIGLAS

ITU-T *Telecommunication Standardization Sector* 16
TPM *Trusted Platform Module* 18
SE *Secure Element* 18
TEE *Trusted Execution Environment* 3
UICC *Universal Integrated Circuit Card* 18
REE *Rich Execution Environment* 19
TA *Trusted Application* 19
USB *Universal Serial Bus* 23
API *Application Programming Interface* 18

SUMÁRIO

1 INTRODUÇÃO . . . 11
1.1 Objetivo Geral . . . 12
1.2 Objetivos específicos . . . 12
1.3 Organização do texto . . . 12

2 FUNDAMENTAÇÃO TEÓRICA . . . 13
2.1 Propriedades básicas de segurança . . . 13
2.2 Criptografia . . . 14
2.2.1 Criptografia simétrica . . . 14
2.2.2 Criptografia assimétrica . . . 14
2.2.3 Comparativo entre criptografia simétrica e assimétrica . . . 17
2.3 Ambientes seguros de execução . . . 17
2.3.1 Trusted Platform Module . . . 18
2.3.2 Secure Element . . . 18
2.3.3 Trusted Execution Environment . . . 19
2.3.4 Comparação entre os ambientes seguros de execução . . . 19
2.4 Atualização de *firmware* . . . 20

3 PROPOSTA . . . 23
3.1 Cronograma das atividades . . . 25

REFERÊNCIAS . . . 27

1 INTRODUÇÃO

Dispositivos que possuem algum tipo de sistema embarcado compartilham um grande espaço na vida das pessoas. De cafeteiras à *smartphones*, estes equipamentos facilitam o cotidiano e proporcionam uma maior comodidade para as pessoas.

Uma vez implantados, estes dispositivos podem apresentar problemas não previstos durante o desenvolvimento, e uma alteração no seu *software* embarcado - também chamado de *firmware* - pode corrigir a falha. Desta forma, os aparelhos devem possuir algum mecanismo de atualização de *firmware*, de preferência de forma remota, que previna ou corrija o mal funcionamento.

A possibilidade de atualizar o *firmware* de um dispositivo de forma remota trás consigo alguns benefícios. Primeiramente, evita que o usuário tenha que retornar o equipamento ao seu fabricante, poupando tempo e dinheiro do consumidor. Além disso, atualizar o *firmware* remotamente provê uma maior agilidade na melhoria dos equipamentos, uma vez que os dispositivos podem ser atualizados assim que o fabricante dos equipamentos disponibilizar a nova versão do *firmware* (JAIN; MALI; KULKARNI, 2016).

A atualização remota de *firmware* pode ainda dar-se de duas formas. Na primeira, o dispositivo detecta que existe uma atualização e aplica a melhoria de forma automática, como é o caso de alguns *smartphones*. Existem também casos onde o usuário do dispositivo deve verificar se existe um novo *firmware* disponibilizado pelo fabricante do equipamento e ele mesmo realizar o procedimento para o *upgrade*, como ocorre com alguns roteadores domésticos ou placas-mãe de computadores (JAIN; MALI; KULKARNI, 2016). Neste trabalho, o foco será em atualização de *firmware* em dispositivos com sistemas embarcados em microcontroladores e a atualização dependerá de uma ação manual do usuário.

Antes de efetivamente atualizar o *firmware* de um dispositivo, deve-se observar propriedades de segurança e confiabilidade do novo *firmware* que será recebido, com o objetivo de garantir que apenas *firmwares* que sejam emitidos por fontes seguras, isto é, fontes conhecidas, possam ser utilizados, além de garantir que o dispositivo não ficará inutilizável após o procedimento de atualização (NIKOLOV, 2018).

Para atingir uma atualização segura, é possível utilizar criptografia de chave pública. Nela, duas chaves distintas são geradas e utilizadas para criptografar e descriptografar uma informação (LEE, 2013). Uma destas chaves deve ser mantida em segredo pelo gerados das chaves e a outra deve ser compartilhada com quem deseja trocar informação com o gerador da chave. A partir disto, o emissor da informação, neste caso, o fornecedor do *firmware*, poderá fazer o uso de assinaturas digitais para cifrar a mensagem com sua chave privada e enviar para o dispositivo que será atualizado. Como em teoria apenas o desenvolvedor do *firmware* possui a chave que cifrou o código, o dispositivo saberá que o *firmware* vem de uma fonte segura.

A propriedade de confiabilidade, por sua vez, poderá ser atingida com a verificação de integridade do *firmware* recebido. Esta verificação pode acontecer a partir do uso de funções de *hash*, que são funções de embaralhamento que geram saídas únicas para cada entrada (STALLINGS, 2008). Deve-se gerar um valor de *hash* tendo como entrada da função o novo *firmware* e enviar para o dispositivo esta informação. Em seguida, o próprio dispositivo deverá realizar o mesmo procedimento, desta vez com o *firmware* que o mesmo recebeu, e comparar o valor gerado com o valor recebido anteriormente. Caso estes valores sejam distintos, houve alguma modificação do *firmware* original para o recebido e desta forma o dispositivo não deverá aceitá-lo.

Diante do que foi posto, neste trabalho propõe-se desenvolver uma solução de atualização de

firmware em dispositivos microcontrolados, de forma que esta solução possua garantias de confiabilidade e segurança da atualização.

1.1 Objetivo Geral

O objetivo deste trabalho é desenvolver uma solução que permita a atualização de *firmware* de um sistema embarcado em um dispositivo microcontrolado, garantindo que a atualização seja realizada de forma segura e confiável.

1.2 Objetivos específicos

Para concluir o objetivo geral, alguns objetivos específicos foram definidos:

1. Definir um conjunto de características fundamentais que um microcontrolador deve prover para a proposta do trabalho;
2. Especificar um procedimento para permitir a transferência e a carga de *firmwares* para o microcontrolador;
3. Especificar procedimentos para garantir a autenticidade dos *firmwares* recebidos pelo microcontrolador;
4. Desenvolver uma prova de conceito da solução;
5. Elaborar um plano de testes para validar a solução desenvolvida.

1.3 Organização do texto

Este trabalho está dividido em duas partes. O Capítulo 2 irá abortar os tópicos julgados necessários para o entendimento e desenvolvimento deste trabalho. Já o Capítulo 3 trás consigo a proposta de desenvolvimento do trabalho, definindo as etapas de desenvolvimento e delimitando seus respectivos prazos para conclusão.

2 FUNDAMENTAÇÃO TEÓRICA

Este capítulo aborda os tópicos julgados relevantes para o desenvolvimento deste trabalho. A seção 2.1 aborda as propriedades básicas de segurança. Já na seção 2.2 são abordadas as criptografias simétricas e assimétricas. A seção 2.3, por sua vez, comenta sobre alguns ambientes seguros de execução baseados em *hardware* e *software*. Por fim, a seção 2.4 aborda alguns pontos necessários para a atualização de *firmwares* de dispositivos com sistemas embarcados.

2.1 Propriedades básicas de segurança

Segundo Mello et al. (2006), segurança é um serviço que garante que um sistema permaneça em execução mesmo quando ações não previstas aconteçam ou usuários não autorizados tentem acessar o sistema, sem que ocorra violação de segurança. De acordo com Russel e Gangemi (1991), existem três diferentes propriedades que sustentam a segurança: confidencialidade, integridade e disponibilidade.

Uma informação não deve ser exposta à usuários que não possuem acesso à mesma. **Confidencialidade** é a garantia de que dados sensíveis sejam mantidos seguros, permitindo que apenas usuários autorizados possam interpretar a informação. Sem a confidencialidade, segredos industriais ou até militares poderiam ser obtidos (RUSSEL; GANGEMI, 1991). A confidencialidade é, portanto, um pilar importante da segurança computacional.

Informações não devem ser corrompidas ou alteradas. A **integridade** garante que o sistema não irá permitir nenhuma modificação não autorizada da informação, seja de forma maliciosa ou acidental (RUSSEL; GANGEMI, 1991). Entretanto, caso haja uma modificação da informação, o sistema deve possibilitar a identificação desta mudança (LEE, 2013).

Um sistema deve estar disponível sempre que requisitado. Desta forma, a **disponibilidade** diz que o acesso de usuários autorizados a um sistema não deve ser negado, mesmo de forma maliciosa (MELLO et al., 2006). Segundo Russel e Gangemi (1991), quando necessário, o sistema deve ser capaz de recuperar seu funcionamento padrão.

Além das já citadas, Landwehr (2001) aborda outras duas propriedades de segurança: autenticidade e não-repúdio. **Autenticidade** é a garantia de que um usuário é quem ele diz ser. Já o **não-repúdio** trás a garantia de que um usuário não poderá afirmar que o mesmo não participou de uma transação

Apesar das propriedades de segurança, ataques de usuários maliciosos podem acontecer, a partir de vulnerabilidades presentes nos sistemas. Bishop e Bailey (1996) definem vulnerabilidade como a caracterização de um estado não autorizado do sistema que pode ser alcançado a partir de um estado autorizado, comprometendo seu funcionamento. De acordo com Shirey (2007), um ataque é um ato intencional o qual um usuário tenta violar as políticas de segurança de um sistema, comprometendo assim as propriedades básicas de segurança. Estes podem ser classificados em duas categorias: ataques passivos e ativos.

Ataques passivos têm a intenção de visualizar a troca de informação entre os pares, sem fazer alguma alteração nas mensagens (STALLINGS, 2008). Um ataque passivo pode acontecer quando uma terceira entidade intercepta uma troca de mensagens e tem acesso à informação obtida.

De acordo com Stallings (2008), ataques ativos fazem modificações nas mensagens interceptadas, permitindo que o atacante tenha o controle da troca de mensagens. Ainda de acordo com Stallings (2008),

ataques ativos podem ser classificados em quatro categorias: disfarce, repetição, modificação de mensagens e negação de serviço.

- Disfarce: Ocorre quando o atacante se faz passar por uma entidade diferente, ou seja, um dos pares comunicantes acha que está conversando com alguém confiável quando na verdade está falando com o atacante;
- Repetição: Quando o atacante captura uma troca de mensagens e posteriormente repassa a mensagem ao receptor, na tentativa de causar algum efeito diferente do esperado pelo sistema;
- Modificação de mensagens: É a alteração de uma mensagem interceptada entre os pares comunicantes, comprometendo a integridade da troca de informações. Ou seja, o par receptor pode receber uma mensagem completamente diferente do que o transmissor enviou, de acordo com as intenções do atacante;
- Negação de serviço: É o impedimento do uso normal das instalações de comunicação, como quando o atacante intercepta todas as mensagens do transmissor e não faz o repasse ao receptor.

2.2 Criptografia

De acordo com Stallings (2008), criptografia é o mapeamento de uma informação (letras, *bits*) em um outro símbolo, de modo que o mapeamento inverso só possa ser feito por um usuário que tenha informações suficientes para realizar a descriptografia. Ao criptografar um dado, é importante que nenhuma informação seja perdida, fazendo com que a mensagem criptografada possa ser revertida.

Se a mesma chave criptográfica for utilizada para transmissão e recepção de uma informação, este sistema é chamado de criptografia simétrica (ou de chave única). Caso chaves diferentes sejam utilizadas na criptografia, este sistema é chamado de criptografia assimétrica (ou de chave pública).

Para o uso correto de criptografia, é necessário que o algoritmo utilizado seja robusto. Na prática, mesmo que o atacante possua várias mensagens criptografadas, ele não deve ser capaz de obter a informação original a partir de padrões característicos do algoritmo utilizado. A informação só deverá ser obtida a partir do uso da chave utilizada durante a cifragem da informação (no caso da criptografia simétrica). Desta forma, não é necessário omitir a informação do algoritmo utilizado. Faz-se necessário apenas proteger a chave criptográfica (STALLINGS, 2008).

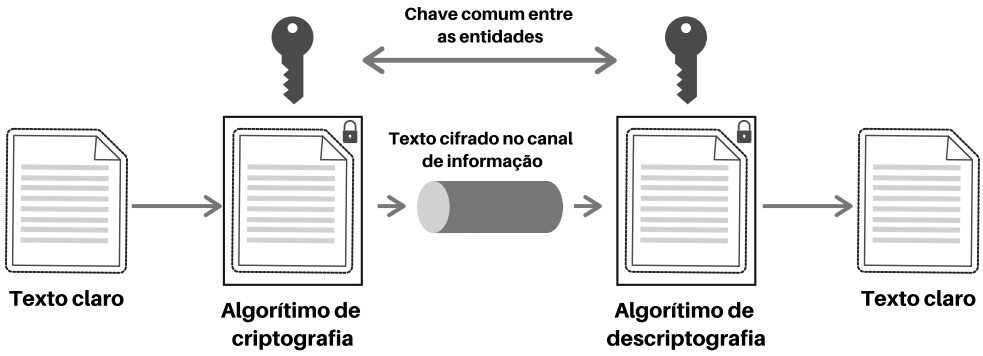
2.2.1 Criptografia simétrica

A criptografia simétrica é aquela onde apenas uma chave é utilizada para criptografar e descriptografar a informação. Desta maneira, a informação a ser criptografada (ou texto claro) é embaralhada a partir do algoritmo de criptografia e a chave secreta. A Figura 1 apresenta o modelo de criptografia simétrica. O algoritmo criptográfico gera saídas diferentes para cada entrada (chave e texto claro) recebida. Portanto, para obter a informação original novamente, é necessário ter informação da chave utilizada (STALLINGS, 2008).

2.2.2 Criptografia assimétrica

Diferentemente da criptografia simétrica, na criptografia assimétrica (ou de chave pública) existem duas chaves distintas para criptografar e descriptografar uma informação: a chave pública e a chave privada do usuário (LEE, 2013). A chave privada deve ser protegida pelo usuário e nunca compartilhada. Já a chave pública deverá ser distribuída entre os outros usuários. Com a criptografia de chave pública é possível

Figura 1 – Modelo de criptografia simétrica



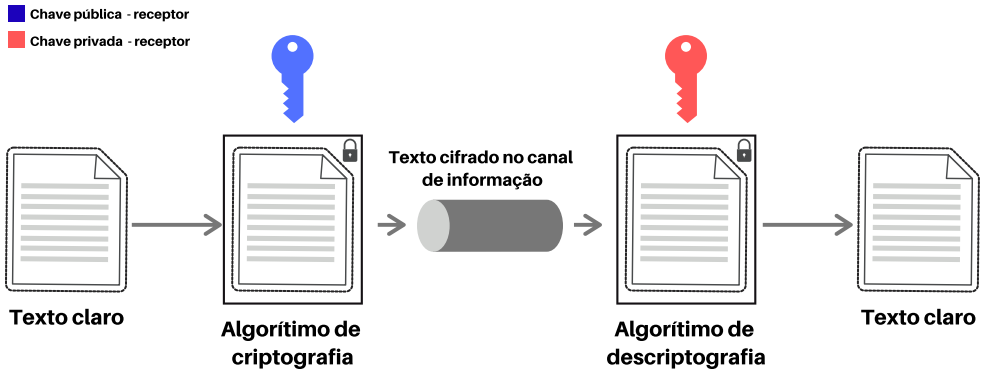
Fonte: Adaptada de (STALLINGS, 2008).

garantir a confidencialidade da informação. A confidencialidade é atingida ao cifrar uma informação com a **chave pública** da entidade destino. Desta forma, é garantido que apenas a chave privada análoga da chave pública utilizada será capaz de descriptografar a mensagem. Este processo pode ser conferido na Figura 2

A partir do uso de chaves privadas, desenvolveu-se as assinaturas digitais. Assinaturas digitais são criadas a partir de funções de *hash*, que possuem como entrada apenas a mensagem a ser embaralhada, podendo ser de tamanho variado. Portanto, não existe uma chave criptográfica nesta função. Sua saída retorna um valor (código de *hash*) de tamanho fixo e não é previsível, isto é, uma pequena mudança na mensagem pode gerar um valor totalmente diferente no código de *hash* (STALLINGS, 2008). A função *hash* é irreversível, não existindo possibilidade de identificar a mensagem que gerou um código *hash* (JOHNER et al., 2000).

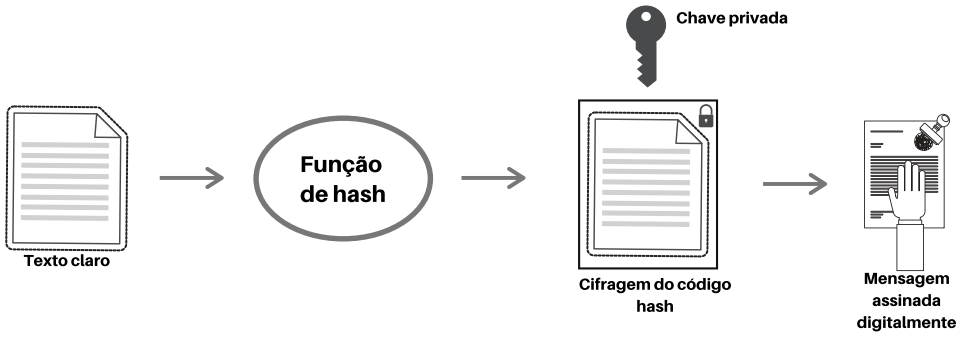
A assinatura digital é o processo de criptografar o código *hash* gerado de uma função *hash*, utilizando a chave privada do emissor. Esta assinatura é então anexada à mensagem a ser transmitida, ficando a cargo do receptor gerar um código *hash* da mensagem recebida, descriptografar a assinatura digital anexada à mensagem com a chave pública do emissor e comparar os dois códigos *hash* obtidos (JOHNER et al., 2000). Enquanto as chaves privadas permanecerem seguras, um usuário não poderá afirmar que o mesmo não assinou uma mensagem outrora assinada por ele, garantindo assim a propriedade do não-repúdio (LEE, 2013). A Figura 3 descreve o processo de assinatura digital de uma mensagem.

Figura 2 – Modelo de criptografia assimétrica



Fonte: Própria.

Figura 3 – Processo de assinatura digital



Fonte: Adaptada de (JOHNER et al., 2000)

Apesar de garantir a integridade da informação recebida, o uso de funções de *hash* e assinaturas digitais não garantem a autenticidade do emissor, uma vez que um atacante pode obter uma chave privada de uma entidade e assumir seu papel em uma troca de informações. Para isto, os certificados digitais foram criados.

Certificado digital é um mecanismo utilizado para prover a autenticidade de uma entidade. Um certificado digital contêm a chave pública de uma entidade e algumas outras informações que possam ajudar a identificar um usuário. Todo certificado digital deve ser assinado por uma entidade, chamada de autoridade certificadora, que é responsável por atestar, através de assinaturas digitais, as identidades dos usuários (JOHNER et al., 2000).

Quando um usuário deseja compartilhar sua chave pública ou trocar informação com outro par, é anexado à mensagem seu certificado digital recebido pela autoridade certificadora. Como esta entidade é confiável por ambas as partes da comunicação, o segundo usuário aceita a chave pública do emissor e ambos podem trocar informações de forma confiável (JOHNER et al., 2000).

Os certificados digitais são criados a partir do formato sugerido na recomendação X.509 da *Telecommunication Standardization Sector* (ITU-T). Segundo Stallings (2008), a recomendação X.509 está na sua terceira versão e tem como base o uso de criptografia de chave pública e assinaturas digitais e prevê no seu modelo de certificado digital onze campos distintos, descritos na Tabela 1:

Tabela 1 – Campos de um certificado digital - recomendação X.509

Campo	**Descrição**
Versão	Variante sendo utilizada da recomendação X.509
Número de série	Um valor inteiro que distingue cada certificado
Identificador do algorítimo	O algoritmo utilizado para assinar digitalmente o certificado
Nome do emissor	Este campo informa o nome da autoridade certificadora que assinou o certificado
Período de validade	As datas de início e fim do período de abrangência do certificado
Nome do titular	O nome do usuário ao qual o certificado digital se refere
Informação da chave pública do titular	É o campo aonde a chave pública do titular do certificado é armazenada
Identificador exclusivo do emissor	Esta identificação distingue autoridades certificadoras que possuem o mesmo nome
Identificador exclusivo do titular	Esta identificação distingue usuários que possuem o mesmo nome.
Extensões	Informações a respeito do certificado, como políticas em que o certificado se aplica e nomes alternativos de emissor e titular
Assinatura	A assinatura contêm os códigos *hash* dos outros campos do certificado, criptografados com a chave privada da autoridade certificadora

Fonte: Própria

Tabela 2 – Comparação entre os tipos de criptografia e as propriedades básicas de segurança

Propriedade	**Criptografia simétrica**	**Criptografia assimétrica**
Confidencialidade	√	√
Integridade		√
Autenticidade		√
Não-repúdio		√
Disponibilidade	não se aplica	não se aplica

Fonte: Própria

2.2.3 Comparativo entre criptografia simétrica e assimétrica

A fim de comparação, a Tabela 2 apresenta quais propriedades básicas de segurança cada tipo de criptografia atende. Ambas as criptografias atendem a propriedade de confidencialidade, uma vez que a informação é cifrada através de algoritmos e chaves criptográficas. Entretanto, apenas a criptografia assimétrica atende a integridade e o não-repúdio, através da assinatura digital, e a autenticidade, através dos certificados digitais.

Também é possível fazer um comparativo entre os tipos de criptografia e os tipos de ataques descritos na seção 2.1. Uma vez criptografada, a mensagem só pode ser lida pelo usuário que possuir a chave de descriptografia, podendo ser a mesma chave utilizada para criptografar a informação ou não, no caso do uso da criptografia simétrica e assimétrica, respectivamente. Já a modificação só pode ser detectada através do uso de assinatura digital, mecanismo que apenas a criptografia assimétrica possui. O disfarce pode ser evitado através de certificados digitais, outro mecanismo que apenas a criptografia de chave pública provê.

Tabela 3 – Comparação entre os tipos de criptografia e ataques

Tipos de ataque	**Criptografia simétrica**	**Criptografia assimétrica**
Ataques passivos	√	√
Disfarce		√
Modificação		√
Negação de serviço	não se aplica	não se aplica
Repetição	não se aplica	não se aplica

Fonte: Própria

2.3 Ambientes seguros de execução

Ambientes seguros de execução são aqueles que proveem armazenamento e execução seguros de aplicações e dados, prevenindo que ataques visando obter ou modificar dados de forma não autorizada sejam bem sucedidos. Os mesmos podem ser desenvolvidos baseados tanto em *software* quanto em *hardware*, o último sendo o foco deste trabalho. De acordo com Bouazzouni, Conchon e Peyrard (2018), as características que definem um ambiente seguro de execução são atingidas através das seguintes premissas:

- Execução isolada: Toda aplicação sensível deve ser executada de forma isolada. Desta forma, aplicações maliciosas não possuem acesso a dados manipulados por uma aplicação ou ao seu código em execução;
- Armazenamento seguro: A integridade e confidencialidade de uma aplicação deve ser mantida. Além disso, deve-se também manter chaves criptográficas, certificados e senhas armazenados de forma segura;

- Adaptação segura: Deve-se prover a integridade e confidencialidade de dados trocados entre aplicações, como chaves criptográficas e certificados.

Dentre as soluções baseadas em *hardware* para alcançar as propriedades de segurança de ambientes seguros de execução, destacam-se três: *Trusted Platform Module* (TPM), *Secure Element* (SE) e *Trusted Execution Environment* (TEE). Estas soluções serão detalhadas nas subseções seguintes.

2.3.1 Trusted Platform Module

Trusted Platform Module (TPM) é um microcontrolador seguro capaz de executar funções criptográficas complexas (BOUAZZOUNI; CONCHON; PEYRARD, 2018). O TPM conecta-se a um outro dispositivo, por exemplo a placa-mãe de um computador de mesa, atuando de forma externa e isolada a mesma, possuindo como uma de suas principais funções a garantia de integridade do dispositivo o qual o TPM está acoplado, guardando informações que podem ser utilizadas, por exemplo, para prover uma inicialização íntegra e segura do sistema (NYMAN; EKBERG; ASOKAN, 2014). As informações entre o TPM e o dispositivo em que ele se conecta-se podem ser trocadas a partir de uma *Application Programming Interface* (API) que o próprio TPM fornece. Segundo Bouazzouni, Conchon e Peyrard (2018), o TPM possui quatro elementos principais:

- Chave de endosso: Consiste em um par de chaves pública e privada. A chave privada é gerada dentro do TPM e nunca utilizada externamente;
- Chaves de atestado de identidade: Estas chaves tem o propósito de atestar a autenticação com um provedor de serviço.;
- Certificados: O TPM guarda três certificados:
  - Certificado de endosso: Este certificado tem a função de garantir a integridade da chave de endosso;
  - Certificado de plataforma: Este certificado é provido pelo fornecedor do módulo e tem a função de garantir que todos os componentes de segurança do TPM são genuínos;
  - Certificado de conformidade: Este certificado pode ser provido por uma terceira parte ou pelo próprio fornecedor do TPM e tem a responsabilidade de garantir que o TPM realmente possui todas as propriedades de segurança que o fabricante informa que o módulo possui.
- Registradores de configuração da plataforma: Estes registradores são utilizados para guardar informações de vínculo entre dados sensíveis a dispositivos ou aplicações.

2.3.2 Secure Element

Secure Element (SE) é um dispositivo inviolável que provê um alto nível de segurança, capaz de armazenar dados sensíveis. Diferentemente do TPM, o SE consegue executar pequenos *softwares* embarcados no dispositivo, e não apenas operações criptográficas (SHEPHERD et al., 2016). Bouazzouni, Conchon e Peyrard (2018) descrevem três tipos de SE: SE embarcado, *Universal Integrated Circuit Card* (UICC) SE e micro SD SE:

- SE embarcado: Neste tipo de solução, o elemento seguro é soldado diretamente com a plataforma onde o mesmo irá atuar. Não existe um padrão a respeito dos direitos de acesso ao SE, cabendo ao fabricante do dispositivo definir que tipo de aplicação os desenvolvedores podem ou não instalar no elemento seguro;

- UICC SE: Estes elementos seguros são amplamente utilizados por empresas de telecomunicações, na forma de um cartão que permite a autenticação em suas respectivas redes. Esta solução prevê certa mobilidade, uma vez que estes elementos seguros não estão soldados junto à um *host*, podendo ser conectado em dispositivos diferentes;

- Micro SD SE: Este tipo de elemento seguro também provê mobilidade para o usuário, uma vez que o mesmo não é soldado na plataforma que irá utilizá-lo. Entretanto, este SE possui o mesmo nível de segurança que os anteriores. Em contraponto com UICC, o Micro SD possui um maior espaço de armazenamento (REVEILHAC; PASQUET, 2009).

2.3.3 Trusted Execution Environment

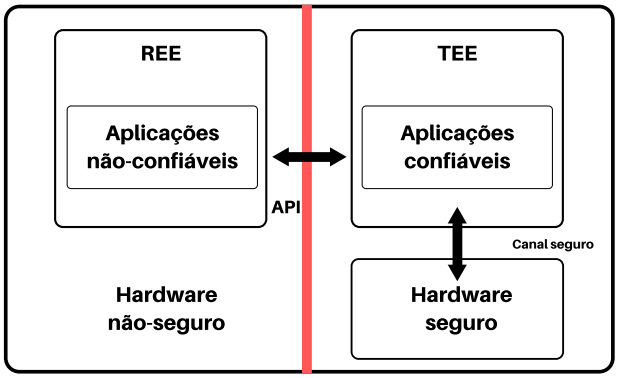
O TEE é um ambiente que possui seu próprio sistema operacional, chamado de sistema operacional seguro. A GlobalPlatform - associação entre empresas para padronização de tecnologias para computação segura (GLOBALPLATFORM, 2018) - especifica o TEE em *software* e *hardware*.

A arquitetura de *hardware* de um sistema que possui um TEE é composta por dois ambientes de execução distintos: *Rich Execution Environment* (REE) e TEE. Estes dois ambientes operam de forma isolada e completamente independentes entre si (ARFAOUI; GHAROUT; TRAORé, 2014).

O REE é o ambiente padrão de um dispositivo, onde aplicações que não necessitam de altos níveis de segurança são executadas, possibilitando uma menor complexidade de implementação. Desta maneira, as aplicações que são executadas no REE não possuem acesso direto às aplicações seguras que estão sendo executadas no outro ambiente (BOUAZZOUNI; CONCHON; PEYRARD, 2018).

O TEE, por sua vez, é um ambiente seguro onde aplicações que precisam de altos níveis de segurança são executadas. O TEE possui um sistema operacional seguro, que é responsável por executar operações sensíveis como funções criptográficas ou capturar dados de entrada de periféricos confiáveis. As aplicações executadas no TEE são chamadas de *Trusted Application* (TA) e a única maneira de aplicações executadas no REE terem acesso às aplicações seguras é a partir de interfaces de aplicação (API) disponibilizadas pelo TEE (BOUAZZOUNI; CONCHON; PEYRARD, 2018). A Figura 4 ilustra de forma simplificada a arquitetura de *software* e *hardware* de um TEE.

Figura 4 – Arquitetura simplificada de um TEE



Fonte: Adaptada de (ARFAOUI; GHAROUT; TRAORé, 2014)

2.3.4 Comparação entre os ambientes seguros de execução

Bouazzouni, Conchon e Peyrard (2018) trazem um comparativo sobre as tecnologias de ambientes seguros de execução, que pode ser observado na Tabela 4. É possível atestar que o TEE possui mais

Tabela 4 – Comparação entre tecnologias de ambientes seguros de execução

Critério	**TPM**	**SE**	**TEE**
Resistência a violações	√	√	
Entrada segura de dados			√
Alta capacidade de processamento	√		√
Alta capacidade de armazenamento			√
Dependência do fabricante	√	√	√

Fonte: Adaptado de (BOUAZZOUNI; CONCHON; PEYRARD, 2018)

limitações físicas quanto a segurança, uma vez que o mesmo não é fisicamente inviolável. Entretanto, o TEE consegue estabelecer um canal de comunicação seguro entre a entrada de dados do dispositivo e as aplicações que estão sendo executadas no mesmo, além de prover maiores capacidades de processamento e armazenamento.

2.4 Atualização de *firmware*

Por inúmeras razões, durante seu ciclo de vida, um dispositivo com um sistema embarcado poderá precisar em algum momento atualizar o seu *software* embarcado, também chamado de *firmware*. Desta forma, faz-se necessário desenvolver mecanismos para que seja possível atualizar o *firmware* de um dispositivo após sua implantação, de preferência de forma remota. Jain, Mali e Kulkarni (2016) defendem que a possibilidade de atualizar o *firmware* de um dispositivo de forma remota trás consigo alguns benefícios, como evitar que o usuário retorne o equipamento ao fabricante para fazer a atualização, bem como facilita e acelera a atualização dos vários dispositivos implantados, uma vez que eles podem ser atualizados no momento em que o fabricante disponibilize o novo código.

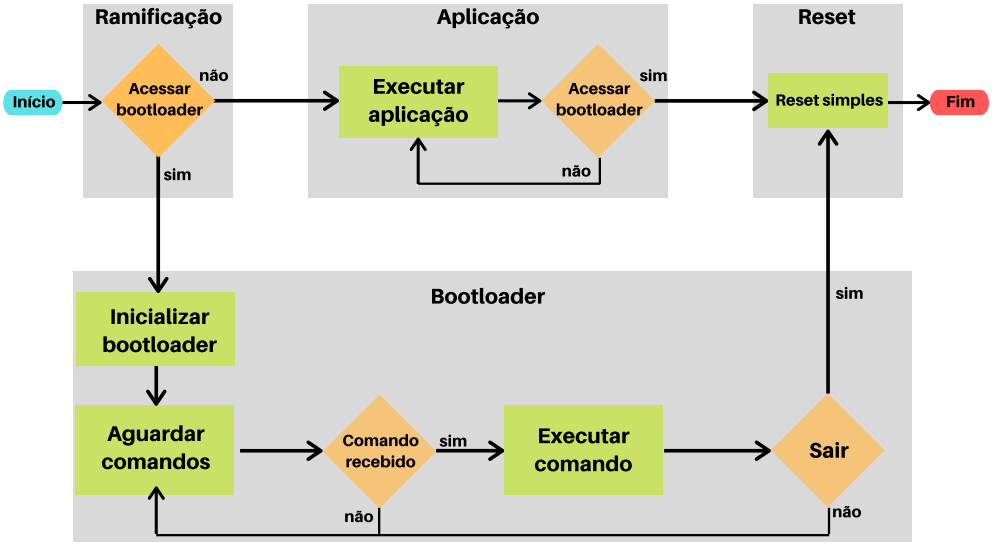
Para que a atualização de um dispositivo possa ser realizada sem a necessidade de um *hardware* externo, uma pequena aplicação - *bootloader* - deve ser embarcada no equipamento. O *bootloader* é uma aplicação que compartilha espaço da memória *flash* de um microcontrolador com o *firmware*, sendo capaz de apagar e escrever novos códigos na área de memória compartilhada, bem como verificar a integridade de um *firmware* (BENINGO, 2015).

O *bootloader* possui dois comportamentos padrão. No primeiro, o processo de execução do *bootloader* é feita de forma totalmente automática e isolada do restante do sistema, o que significa dizer que a detecção de um novo *firmware* e sua posterior instalação é feita de forma autônoma. No segundo modelo, o sistema não executa o *bootloader* de forma autônoma. Ao invés disto, o sistema entra em um estado ocioso e aguardará os comando para a atualização do *firmware* de uma fonte externa ao sistema (JAIN; MALI; KULKARNI, 2016).

Beningo (2015) afirma que cada projeto de *bootloader* irá possuir seus requisitos particulares necessários. Todavia, existem alguns requisitos fundamentais que todo projeto de *bootloader* deve possuir:

1. Habilidade de selecionar os modos de operação do dispositivo (*bootloader* ou aplicação);
2. Interfaces de comunicação com outros dispositivos;
3. Padrão definido quanto ao formato de arquivo do código do *firmware*;
4. Possibilidade de manipulação das memórias do sistema;
5. Verificação de integridade do *firmware*;
6. Armazenamento do *bootloader* de forma que seu código não possa ser modificado;

Figura 5 – Componentes de um sistema com *bootloader*



Fonte: Adaptada de (BENINGO, 2015)

Em geral, um sistema embarcado deve possui três componentes principais: o código de ramificação, a aplicação (*firmware*) e o *bootloader*. O código de ramificação é responsável por informar se o sistema deve executar a aplicação do sistema ou o *bootloader*. A aplicação é executada quando o código de ramificação define que o sistema não deve executar o *bootloader*. Em certos casos, como quando a integridade da aplicação deve ser verificada na inicialização do dispositivo, o código de ramificação e o *bootloader* são uma coisa só. Deve existir também um bloco de reinicialização do sistema (BENINGO, 2015). A Figura 5 ilustra os componentes do sistema, juntamente com um comportamento padrão do mesmo.

Quando selecionado, o *bootloader* entrará em execução, devendo prover suporte para pelo menos três comandos:

- Remover o *firmware* da memória *flash*;
- Escrever na memória *flash* um novo *firmware*;
- Sair do *bootloader* e carregar o *firmware* instalado.

Alguns outros comandos devem ser previstos, de acordo com os requisitos fundamentais já descritos, a fim de melhorar as capacidades de atualização de *firmware* do *bootloader*, tais como bloquear e desbloquear a manipulação da memória *flash* e gerar um valor a partir de uma função de *hash* que garanta

Figura 6 – Processo padrão de atualização de *firmware*

Fonte: Adaptada de (BENINGO, 2015)

a integridade do *firmware* instalado. Beningo (2015) descreve uma sequência padrão para a atualização de *firmware* de um sistema, que pode ser observada na Figura 6.

3 PROPOSTA

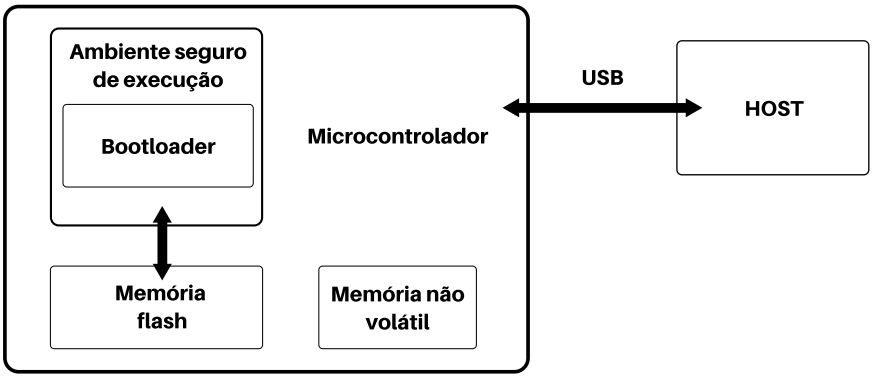
A proposta deste trabalho é desenvolver uma solução de atualização de *firmware* em sistemas embarcados microcontrolados, de forma segura e confiável. Neste contexto, entende-se como atualização segura uma solução que garanta que apenas *firmwares* emitidos por fontes fidedignas possam ser recebidos e utilizados pelo dispositivo. Uma atualização confiável, por sua vez, é aquela que garanta que o dispositivo não ficará inutilizável após o processo de atualização, mesmo que a integridade do *firmware* seja comprometida.

Para atingir a propriedade de **segurança**, os *firmwares* deverão ser assinados digitalmente pelo desenvolvedor do dispositivo, fazendo o uso da criptografia de chave pública para atestar a autenticidade de quem emitiu o *firmware*. Já a **confiabilidade** será atingida com a verificação da integridade do *firmware* inserido no dispositivo, a partir do uso de funções *hash*. Após a inserção do novo *firmware* no microcontrolador, será obtido o valor de *hash* do mesmo e este valor deverá ser comparado com o valor de *hash* que o servidor também deverá enviar. Desta forma é possível verificar se ambos os valores são iguais, atestando a integridade do *firmware* recebido. Uma vez verificada a integridade do *firmware*, o valor de *hash* deve ser armazenado em memória, devendo ser usado para atestar a integridade do dispositivo a cada inicialização.

Propõe-se armazenar o *bootloader* do dispositivo em um ambiente seguro de execução baseado em *hardware*, como os ambientes citados na seção 2.3. O dispositivo será conectado a um *host* por meio de uma *Universal Serial Bus* (USB), pela qual a troca de dados irá ser realizada. Uma versão simplificada dos componentes do dispositivo proposto pode ser observada na Figura 7. Além do dispositivo a ser atualizado e o *host*, também é previsto neste trabalho um servidor de *firmwares*. Caberá ao *host* verificar se existem *firmwares* novos no servidor e informar ao dispositivo para entrar em modo de atualização de *firmware*, bem como enviar os *firmwares* novos para o mesmo. Já o servidor de *firmwares* será responsável por armazenar os novos *firmwares* desenvolvidos, que deverão possuir uma assinatura digital válida.

A Figura 8 apresenta um diagrama de atividades UML que ilustra a interação entre as entidades. Nela, a partir de uma ação do usuário, o *host* faz uma requisição ao servidor de *firmwares* a fim de verificar a existência de novas versões. Uma vez que exista uma resposta afirmativa do servidor, o *host* deverá receber o novo *firmware* e informar ao dispositivo que o mesmo entrará em processo de atualização, passando o controle da operação ao *bootloader*. Uma vez em processo de atualização, o dispositivo deverá

Figura 7 – Componentes do dispositivo proposto.

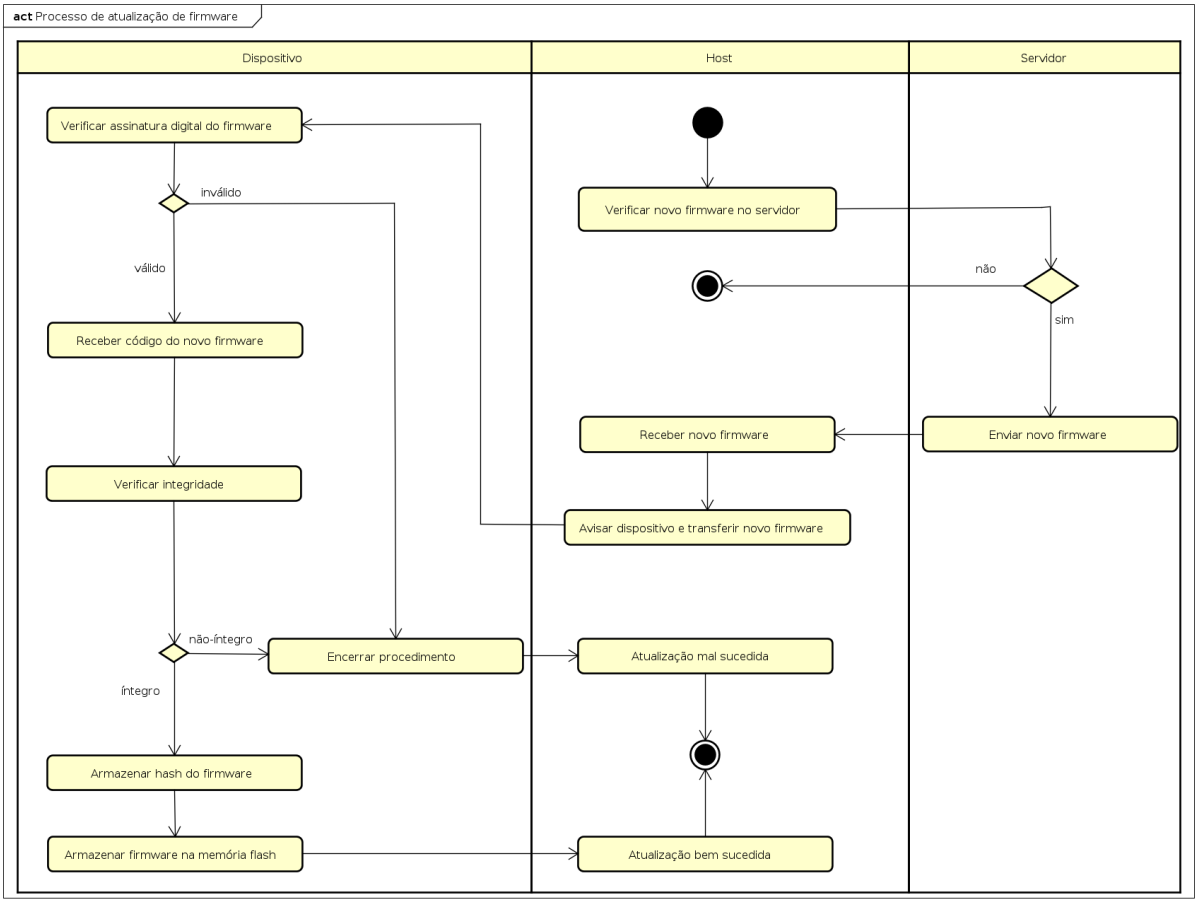


Fonte: Própria.

verificar se o *firmware* recebido vem de uma fonte confiável, avaliando a assinatura digital do *firmware*. Na hipótese de a assinatura ser válida, o *bootloader* poderá limpar as áreas de memória do dispositivo necessárias para o armazenamento temporário do código e receber o novo *firmware*. Após este processo, deverá ser verificado a integridade do código recebido, utilizando funções de *hash* para obter um valor de *hash* do *firmware* que chegou ao dispositivo e comparar com um valor de *hash* que deverá vir anexado ao *firmware* provido pelo servidor. Na prática, caso os dois valores de *hash* coincidam, o *firmware* não sofreu alterações e sua integridade foi confirmada. Esta informação da integridade do *firmware* pode ser armazenada em memória para que seja verificada sempre que o dispositivo for iniciado. Sabendo que o *firmware* é íntegro, será possível armazená-lo na memória *flash* do dispositivo e o processo de atualização poderá ser finalizado.

Para validar a solução a ser implementada, alguns testes deverão ser feitos. Dentre os testes que serão realizados, destacam-se os testes que verificarão as propriedades de segurança e confiabilidade, tais como: envio de novos *firmwares* sem assinatura digital ou assinatura inválida, envio de *firmwares* não íntegros e cerceamento de energia durante o processo de atualização, desconectando o dispositivo do *host*. Estes testes deverão aumentar a certeza de que o dispositivo nunca ficará inutilizável durante ou após a atualização, bem como garantirão que *firmwares* não seguros não serão instalados no dispositivo.

Figura 8 – Diagrama de atividade da proposta para atualização de *firmware*.



Fonte: Própria.

3.1 Cronograma das atividades

É previsto neste trabalho cinco atividades que deverão ser desenvolvidas ao longo de sete meses. Estas atividades estão descritas abaixo, bem como estão planejadas de acordo com a Tabela 5

Tabela 5 – Cronograma de atividades

Atividades	**Mês**						
	Jan	**Fev**	**Mar**	**Abr**	**Mai**	**Jun**	**Jul**
A1	√						
A2		√	√				
A3			√	√	√	√	
A4					√	√	
A5			√	√	√	√	√

- A1: Estudo e definição das características de um microcontrolador que atenda as necessidades do trabalho;
- A2: Definição de procedimentos da atualização que garantam a confiabilidade e a segurança;
- A3: Desenvolvimento da prova de conceito da solução;
- A4: Elaboração de plano de testes para validação da implementação;
- A5: Documentação do desenvolvimento.

REFERÊNCIAS

ARFAOUI, G.; GHAROUT, S.; TRAORé, J. Trusted execution environments: A look under the hood. In: *2014 2nd IEEE International Conference on Mobile Cloud Computing, Services, and Engineering*. [S.l.: s.n.], 2014. p. 259–266. Citado na página 19.

BENINGO, J. Bootloader design for microcontrollers in embedded systems. In: . [s.n.], 2015. Disponível em: <https://www.beningo.com/wp-content/uploads/images/Papers/bootloader_design_for_microcontrollers_in_embedded_systems%20.pdf>. Citado 3 vezes nas páginas 20, 21 e 22.

BISHOP, M.; BAILEY, D. *A Critical Analysis of Vulnerability Taxonomies*. [S.l.], 1996. Citado na página 13.

BOUAZZOUNI, M. A.; CONCHON, E.; PEYRARD, F. Trusted mobile computing: An overview of existing solutions. *Future Generation Computer Systems*, v. 80, p. 596 – 612, 2018. Disponível em: <http://www.sciencedirect.com/science/article/pii/S0167739X16301510>. Citado 4 vezes nas páginas 17, 18, 19 e 20.

GLOBALPLATFORM. Trusted execution environments (tee): An introduction to tee functionality and how globalplatform supports it. In: _____. *Introduction to trusted execution environments*. 2018. Disponível em: <https://globalplatform.org/wp-content/uploads/2018/05/Introduction-to-Trusted-Execution-Environment-15May2018.pdf>. Acesso em: 31 ago. 2019. Citado na página 19.

JAIN, N.; MALI, S. G.; KULKARNI, S. Infield firmware update: Challenges and solutions. In: *2016 International Conference on Communication and Signal Processing (ICCSP)*. [S.l.: s.n.], 2016. p. 1232–1236. Citado 2 vezes nas páginas 11 e 20.

JOHNER, H. et al. *Deploying a Public Key Infrastructure*. [S.l.]: IBM, 2000. ISBN 9780738415734. Citado 2 vezes nas páginas 15 e 16.

LANDWEHR, C. Computer security. *International Journal of Information Security*, v. 1, p. 3–13, 2001. Citado na página 13.

LEE, R. B. *Security Basics for Computer Architects*. [S.l.]: Morgan & Claypool Publishers, 2013. ISBN 9781627051569. Citado 4 vezes nas páginas 11, 13, 14 e 15.

MELLO, E. R. de et al. Segurança em serviços web. In: *Minicursos do Simpósio Brasileiro em Segurança da Informação e de Sistemas Computacionais - SBSeg 2006*. [s.n.], 2006. Disponível em: <http://docente.ifsc.edu.br/mello/artigos/mellomcsbseg06.pdf>. Citado na página 13.

NIKOLOV, N. Research firmware update over the air from the cloud. In: *2018 IEEE XXVII International Scientific Conference Electronics - ET*. [S.l.: s.n.], 2018. p. 1–4. ISSN null. Citado na página 11.

NYMAN, T.; EKBERG, J.-E.; ASOKAN, N. Citizen electronic identities using tpm 2.0. In: *Proceedings of the 4th International Workshop on Trustworthy Embedded Devices*. New York, NY, USA: ACM, 2014. (TrustED '14), p. 37–48. ISBN 978-1-4503-3149-4. Disponível em: <http://doi-acm-org.ez130.periodicos.capes.gov.br/10.1145/2666141.2666146>. Citado na página 18.

REVEILHAC, M.; PASQUET, M. Promising secure element alternatives for nfc technology. In: *2009 First International Workshop on Near Field Communication*. [S.l.: s.n.], 2009. p. 75–80. ISSN null. Citado na página 19.

RUSSEL, D.; GANGEMI, G. T. *Computer Security Basics*. [S.l.]: O'Reilly & Associates, 1991. ISBN 0-937175-71-4. Citado na página 13.

SHEPHERD, C. et al. Secure and trusted execution: Past, present and future – a critical review in the context of the internet of things and cyber-physical systems. In: . [S.l.: s.n.], 2016. Citado na página 18.

SHIREY, R. W. *Internet Security Glossary, Version 2*. RFC Editor, 2007. RFC 4949. (Request for Comments, 4949). Disponível em: <https://rfc-editor.org/rfc/rfc4949.txt>. Citado na página 13.

STALLINGS, W. *Criptografia E Segurança De Redes - Princípios e Práticas*. PEARSON BRASIL, 2008. ISBN 9788543005898. Disponível em: <https://books.google.com.br/books?id=KO8gvgAACAAJ>. Citado 5 vezes nas páginas 11, 13, 14, 15 e 16.