Pós-Graduação em Ciência da Computação

Gibson Belarmino Nunes Barbosa

SISTEMA DE SEGURANÇA PARA IOT BASEADO EM AGRUPAMENTO DE SMART CARDS GERENCIADOS POR FPGA

Universidade Federal de Pernambuco
posgraduacao@cin.ufpe.br
www.cin.ufpe.br/~posgraduacao

RECIFE
2017

Gibson Belarmino Nunes Barbosa

SISTEMA DE SEGURANÇA PARA IOT BASEADO EM AGRUPAMENTO DE SMART CARDS GERENCIADOS POR FPGA

Trabalho apresentado ao Programa de Pós-graduação em Ciência da Computação do Centro de Informática da Universidade Federal de Pernambuco como requisito parcial para obtenção do grau de Mestre em Ciência da Computação.

Orientador: *Djamel Fawzi Hadj Sadok*
Co-Orientadora: *Patricia Takako Endo*

RECIFE
2017

Catalogação na fonte
Bibliotecária Monick Raquel Silvestre da S. Portes, CRB4-1217

B238s Barbosa, Gibson Belarmino Nunes
Sistema de segurança para IoT baseado em agrupamento de smart cards gerenciados por FPGA / Gibson Belarmino Nunes Barbosa. – 2017.
85 f.: il., fig., tab.

Orientador: Djamel Fawzi Hadj Sadok.
Dissertação (Mestrado) – Universidade Federal de Pernambuco. CIn, Ciência da Computação, Recife, 2017.
Inclui referências.

1. Redes de computadores. 2. Internet das coisas. I. Sadok, Djamel Fawzi Hadj (orientador). II. Título.

004.6 CDD (23. ed.) UFPE- MEI 2017-216

Gibson Belarmino Nunes Barbosa

Sistema de Segurança para IoT Baseado em Agrupamento de Smart Cards Gerenciados por FPGA

Dissertação de Mestrado apresentada ao Programa de Pós-Graduação em Ciência da Computação da Universidade Federal de Pernambuco, como requisito parcial para a obtenção do título de Mestre em Ciência da Computação

Aprovado em: 31/08/2017.

BANCA EXAMINADORA

_____

Prof. Dr.Abel Guilhermino da Silva Filho
Centro de Informática / UFPE

_____

Prof. Dr. Danilo Monteiro Ribeiro
Departamento de Estatística e Informática / UFRPE

_____

Prof. Dr. Djamel Fawzi Hadj Sadok
Centro de Informática / UFPE
(Orientador)

Amemos a Deus, meus irmãos, amemos a Deus, mas que seja à custa de nossos braços, que isto seja com o suor dos nossos rostos.

—SÃO VICENTE DE PAULO

RESUMO

Os crescentes casos de ataques a dispositivos conectados às redes de Internet das Coisas (*do inglês Internet of Things - IoT*), com invasores se aproveitando do baixo poder de processamento e segurança frágil desses equipamentos, são fatores determinantes para as atenções serem voltadas à segurança em IoT. A utilização de Cartões Inteligentes (*do inglês Smart Cards - SCs*) para proteger esses sistemas é pertinente, pois eles são resistentes a ataques físicos, portáteis e baratos. Porém, o processamento e a memória deles ainda não são suficientes para atender a alta demanda dos dispositivo conectados à IoT. Por estas razões, utilizar vários SCs agrupados pode ser interessante para proteger tais redes de dispositivos IoT. Contudo, o gerenciamento de muitos SCs requer uma plataforma que não degrade o desempenho do processo de leitura e escrita. Os Arranjos de Portas Programáveis em Campo (*do inglês Field Programmable Gate Arrays - FPGAs*) se encaixam nesse critério, pois são dispositivos reconfiguráveis com capacidade de paralelização de tarefas e replicação de código. Associando esses dispositivos, e conectando-os à rede, seria possível agregar segurança a outros sistemas e garantir a integridade da comunicação na IoT. Estes fatores motivaram o presente trabalho a propor e desenvolver o *Smart Card Cluster* (SCC), que é uma plataforma de segurança para IoT. Ele pode ser empregado de diversas formas para oferecer proteção às camadas de rede ou de aplicação da IoT. Contudo, neste trabalho ele será utilizado para agregar proteção às mensagens que são trocadas entre *gateways* e o *middleware* IoT, assegurando a autenticidade e integridade das mensagens por meio de assinatura feita em hardware seguro. Com três SCs o SCC permite assinar 26,18 mensagens por minuto. O sistema utiliza FPGA para fazer o controle eficiente de SCs. Através de experimentos concluiu-se que ele é modular, portátil, escalável e com custo adequado ao cenário de IoT ao qual é aplicado. Isto possibilita que o utilizador evite gastos, pois os dispositivos embarcados, o poder de processamento, e a quantidade de SCs podem ser ajustados de acordo com a necessidade. Além do que, a plataforma pode ser distribuída para perto dos dispositivos na ponta da rede. No trabalho é mostrado que o SCC consegue ser 69% mais eficiente que outro dispositivo comercial utilizado para agrupamento de SCs, que é conhecido como SIM Array, além disso o custo da utilização de cada SC no SCC chega a ser US$263,54 menor.

Palavras-chave: Assinatura Digital. FPGA. Grid de Elementos Seguros. Internet das Coisas. Segurança. Smart Cards.

ABSTRACT

The increasing cases of attacks on devices connected to *Internet of Things* (IoT) networks, where attackers take advantage of the low processing power and the fragile protection of such equipments, is a determining factor for giving greater attention to IoT security. The use of *Smart Card*s (SCs) to protect these systems is pertinent, as they are physical attacks resistant, portable and inexpensive. Nonetheless, their processing and memory are still not sufficient to meet the high demand of the devices connected to IoT. For these reasons, using multiple clustered SCs may be interesting to protect such networks. However, a platform that does not degrade the performance of the read and write process, is needed to control many SCs. The Field Programmable Gate Arrays (FPGAs) fit this criterion, since they are reconfigurable devices with the ability to parallelize tasks and code replication. By connecting these devices to the network, it would be possible to add security to other systems and ensure communication integrity for the IoT. These factors led to the development of the Smart Card Cluster (SCC), which is a platform for IoT security. It can be used in a variety of ways to protect the IoT layers or applications. In this work, it is used to add protection to messages that are exchanged between gateways and middleware IoT, ensuring message authenticity and integrity through secure hardware signing. With three SCs the SCC allows to sign 26.18 messages per minute. The system uses FPGA to efficiently manage SCs. It is modular, portable, scalable and cost-effective to user's needs. This enables the user to avoid unnecessary costs, since the embedded devices, processing power, and the amount of SCs are adapted to the application scenario. Thus, the platform can be distributed close to the things. In this work it is shown that the SCC becomes 69% more efficient than another commercial device used for clustering SCs, the SIM Array, moreover the cost of using each SC in the SCC is US$ 263.54 lower.

Keywords: Digital Signature. FPGA. Grid of Secure Elements. Internet of Things. Security. Smart Cards.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ACRÔNIMOS

APDU	*Application Protocol Data Unit*
API	*Application Programming Interface*
ASIC	*Application-Specific Integratted Circuit*
ATR	*Answer To Reset*
BGT	*Block Guard Time*
BWT	*Block Wait Time*
CCID	*Circuit(s) Cards Interface Device*
CGT	*Character Guard Time*
CoSE	*Cloud of Secure Elements*
CPLD	*Complex Programmable Logic Device*
CRC	*Cyclic Redundancy Check*
CWT	*Character Wait Time*
DDoS	*Distributed Denial of Service*
DK	*Development Kit*
DSP	*Digital Signal Processing*
LE	*Logic Element*
ETU	*Elementary Time Unit*
FIFO	*First In, First Out*
FPGA	*Field Programmable Gate Array*
GoSE	*Grid of Secure Elements*
GPIO	*General Purpose Input/Output*
HDL	*Hardware Description Language*
ICC	*Integrated Circuit Card*
IFD	*Interface Device*
I/O	*Input/Output*
IoT	*Internet of Things*
I2C	*Inter-Integrated Circuit*

JSON	*JavaScript Object Notation*
LRC	*Longitudinal Redundancy Check*
MQTT	*Message Queue Telemetry Transport*
NRE	*Non-Recurring Engineering*
PC/SC	*Personal Computer/Smart Card*
PKI	*Public Key Infrastructure*
P/S	*Publish/Subscribe*
SC	*Smart Card*
SCC	*Smart Card Cluster*
SES	Servidor de Elementos Seguros
SO	Sistema Operacional
TPDU	*Transport Protocol Data Unit*
USB	*Universal Serial Bus*

SUMÁRIO

1 INTRODUÇÃO

O tempo de projeto, as tecnologias associadas e a mão de obra disponível são exemplos de restrições que devem ser superadas no desenvolvimento de um sistema computacional. A priorização desses fatores pode causar o negligenciamento da proteção do sistema contra ataques, que fica sendo postergada para versões futuras do sistema. Isso viabiliza às pessoas maliciosas tirarem o sistema do ar, alterarem alguma informação ou roubarem dados confidenciais dos usuários. O resultado é um sistema inseguro, pois, considerando a definição de [45], dentre os princípios que são necessários para um sistema ser considerado seguro estão a confidencialidade, para que os dados não sejam visualizados sem autorização, a integridade, para garantir que as informações não sejam modificadas, e a disponibilidade, para que os dados estejam acessíveis no momento em que se precisa deles.

Considerando a Internet das Coisas (*do inglês Internet of Things - IoT*), em 2017 haverá 8.4 bilhões de equipamentos ativos e pouco mais de 20 bilhões até 2020, segundo projeção feita por [25]. Com tantos dispositivos conectados, muitas vulnerabilidades vão surgindo. Há casos de babás eletrônicas sendo acessadas por estranhos [57], *hackers* conseguindo controlar carros remotamente [26], e utilização de câmeras IP para ataques Distribuídos por Negação de Serviço (*do inglês Distributed Denial of Service - DDoS*) [22]. Com todos esses problemas ocorrendo, as questões de segurança em tais aparelhos não podem mais ser procrastinadas.

A pesquisa conduzida por [67] traz as tendências em IoT, coletadas a partir da opinião de 713 desenvolvedores que trabalham com essa tecnologia. Na Figura 1.1, retirada de tal estudo, é possível ver que a segurança é uma das maiores preocupações entre as pessoas desse meio. [94] mostra que, entre os desafios encontrados na área de segurança em IoT, é necessária a discussão sobre integridade e confidencialidade dos dados enviados, assim como a autenticidade dos dispositivos incluídos numa rede.

Para [28] a melhor forma de agregar segurança à IoT é utilizando estratégias de hardware, uma vez que eles têm tempo de vida maior e necessitam de menos atualizações que os softwares. Os hardwares a serem utilizados seriam os Cartões Inteligentes (*do inglês Smart Cards - SCs*), que são empregados como sistema de segurança em cartões de crédito, passaportes, *smartphones*, entre várias outras aplicações. As principais vantagens que eles trazem são a segurança, a

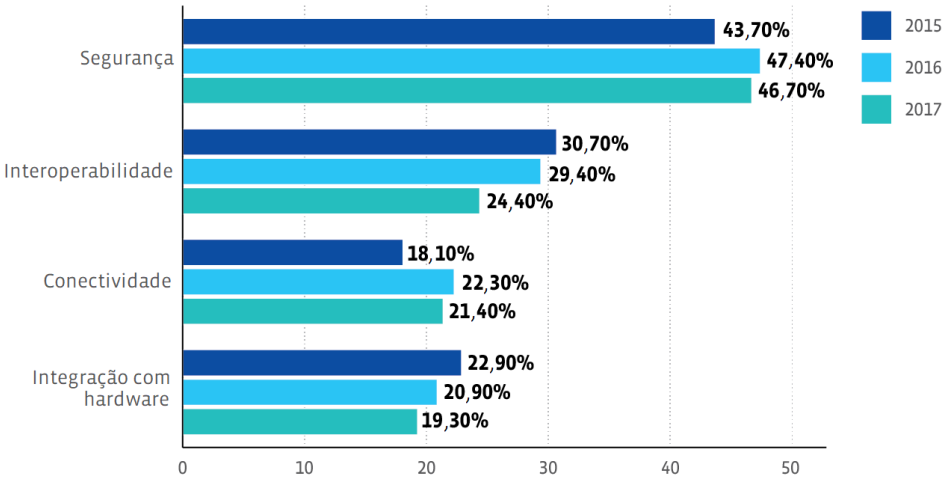


Figura 1.1: Principais preocupações em IoT. Imagem modificada de [67].

portabilidade e o baixo custo.

O grande número de dispositivos conectados à IoT, resulta em uma imensa quantidade de dados trafegando em tais redes. Utilizar SC para inserir segurança às mensagem que trafegam nessas redes é uma ideia interessante, já que eles podem encriptar e assinar dados internamente, e são resistentes a ataques físicos [59]. Porém, as suas restrições de processamento e capacidade de armazenamento, não permitem que apenas um SC atenda a demanda da IoT. Agrupar vários SCs pode solucionar o problema anterior, mas o dispositivo que faz o gerenciamento dessa quantidade crescente de SCs não pode ser um limitador na transmissão dos dados, nem ser financeiramente inviável. Empregar Arranjos de Portas Programáveis em Campo (*do inglês Field Programmable Gate Arrays - FPGAs*) nessas circunstâncias é pertinente, já que eles oferecem processamento em paralelo e muitas interfaces de Entrada e Saída (*do inglês Input/Output - I/O*). Uma plataforma que possa utilizar diferentes tipos de FPGAs, e que além de possibilitar que um FPGA gerencie quantidades variáveis de SCs, permita que seja expandido o número de FPGAs, acrescenta portabilidade e escalabilidade, com baixo custo, já que estes dispositivos podem ser escolhidos de acordo com os equipamentos disponíveis e nível de segurança exigido. Essas vantagens reunidas, permitem que a plataforma de segurança funcione de forma distribuída, trazendo a proteção para mais perto dos dispositivos na ponta da IoT.

O SCC é a plataforma de segurança para IoT que traz as funcionalidades citadas anteriormente. Ele será proposto e descrito nesse trabalho, sendo utilizado de forma distribuída para agregar segurança às mensagens que trafegam entre grupos de dispositivos, associados por um *gateway*, e o *middleware* IoT. Como *middlewares* são programas utilizados para intermediar a troca de informações provenientes de diferentes aplicações, podemos considerar *middlewares* IoT aqueles que são aplicados para fazer o intermédio entre sistemas que estão no contexto de IoT.

1.1 Contextualização

No contexto de cidades inteligentes há uma formação natural de grupos de atuação, seja para casas inteligentes, *smart grids*, assistência médica inteligente, monitoramento ambiental, entre outros. Assim, a heterogeneidade de dispositivos conectados e informações trafegando em redes diversas criam uma série de vulnerabilidades que devem ser tratadas, como mostra [93]. Uma abordagem para resolver isto seria utilizar uma plataforma que agrupe dispositivos seguros, com o intuito de coordenar a segurança para diferentes grupos, de forma centralizada, como na Figura 1.2. Um SC seria responsável pela segurança dos carros inteligentes, outro se aplicaria ao monitoramento ambiental, e para cada um dos outros grupos teria-se um SC. Só seriam habilitados a utilizar a rede, os dispositivos que estivessem pré-configurados no SC responsável por aquele grupo de atuação. Ou seja, para um carro se conectar ao sistema de segurança, sua identificação deveria ter sido previamente inserida no SC que controla a segurança dos carros.

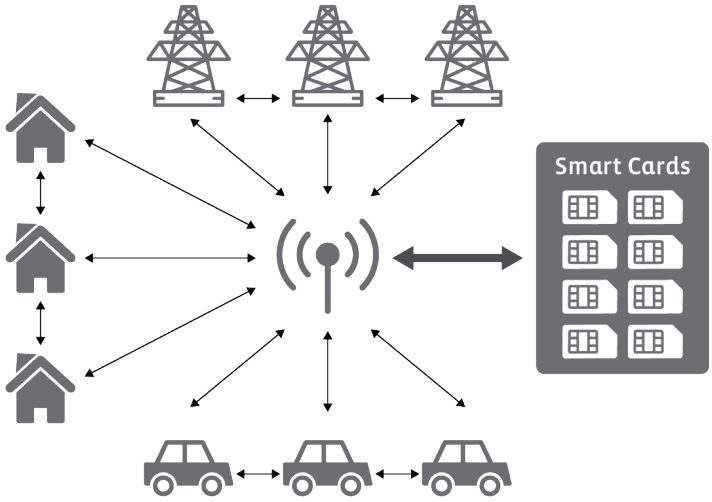


Figura 1.2: Grupos de dispositivos em uma cidade inteligente.

As empresas possuem diversos equipamentos que podem ser agrupados, como condicionadores de ar, câmeras de segurança e medidores inteligentes, como mostra a Figura 1.3. A associação de tais dispositivos pode ocorrer de acordo com o protocolo de transmissão, com o formato das mensagens ou com os dados a serem enviados. Vinculando um SC aos aparelhos pertencentes a um mesmo conjunto, ou aos *gateways*, seria possível agregar processos de controle de acesso e autenticação ao grupo. É até admissível expandir essa aplicação à criptografia das mensagens desse grupo. O conceito de autenticação baseado em grupos proposto primeiramente por [17] e avaliado por [69], traz uma melhoria significativa na performance do processo de autenticação para dispositivos em IoT.

Como os SCs são desenvolvidos para ser portáteis e não têm muita capacidade de armazenamento, a aplicação de um dispositivo que agrupe vários SCs possibilitaria que mais dados pudessem ser arquivados, o que permitiria o armazenamento seguro de dados sob demanda.

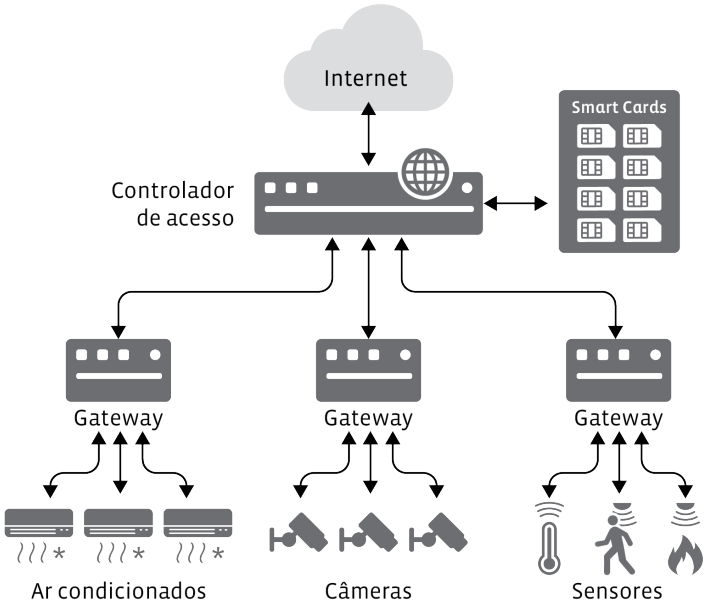


Figura 1.3: Dispositivos conectados em uma empresa.

[73] já demonstrou a utilização de SCs para armazenamento seguro de documentos digitais. Os SCs também pode ser aplicados para autenticação de serviços de Voz sobre IP (VoIP). E, para testar SIM *cards* em larga escala, sem a necessidade de aplicar vários celulares nestes testes.

1.2 Objetivos

O principal objetivo desta dissertação de mestrado é propor e avaliar uma plataforma de segurança para IoT, utilizando SCs gerenciados por FPGA. Esse sistema deve possuir as seguintes características: portabilidade, escalabilidade, modularidade, compatibilidade e baixo custo. Tais princípios tornam viável distribuí-lo entre diferentes grupos de dispositivos associados por um *gateway*. Isso permite que as mensagens trocadas entre os *gateways* e o *middleware* IoT da rede mantenham a integridade e a autenticidade, que são alcançadas através da assinatura por meio do hardware seguro.

Objetivos específicos:

1. Avaliar o impacto da aplicação de diferentes tipos de FPGAs no consumo de recursos e custo do sistema.

2. Comparar o desempenho e o custo do sistema proposto com outra plataforma de agrupamento de SCs.

3. Analisar o efeito do agrupamento de diferentes quantidades de SCs no desempenho do sistema.

4. Fazer um estudo de caso do sistema aplicado em uma arquitetura real de IoT.

1.3 Organização do Trabalho

O presente trabalho está organizado da seguinte forma:

- **Capítulo 2 - Fundamentação teórica:** descreve os dispositivos, protocolos e segurança em IoT, para uma melhor entendimento do trabalho proposto.
- **Capítulo 3 - Estado da arte:** a literatura referente a segurança em IoT é revisada. São relacionados trabalhos que propõem leitores de SCs utilizando FPGA e trabalhos que utilizam dispositivos para agrupamento de SCs.
- **Capítulo 4 - O *Smart Card Cluster*:** a plataforma proposta é detalhada, desde a arquitetura interna até o cenário de aplicação.
- **Capítulo 5 - Resultados e estudo de caso:** o sistema é avaliado e os resultados são discutidos.
- **Capítulo 6 - Conclusão:** as conclusões acerca do trabalho, os problemas enfrentados e possíveis trabalhos futuros são relatados.

2

FUNDAMENTAÇÃO TEÓRICA

Neste capítulo serão detalhados os tópicos relevantes evidenciados no trabalho. Na Seção 2.1 são feitas algumas considerações sobre IoT. A Seção 2.2 retrata questões relacionadas a segurança de sistema. A Seção 2.3 relata os dispositivos embarcados utilizados: SCs, leitores de SCs, FPGAs e servidores de elementos seguros. Por fim, Na Seção 2.4 há a descrição dos protocolos utilizados: ISO/IEC 7816-3, CCID e PC/SC.

2.1 IoT

Para [50], IoT é a combinação de vários objetos físicos (ou coisas), aos quais se acrescentam eletrônica, software e conectividade de rede, para permitir que haja a coleta e troca de mensagens entre eles e outros nós na rede. A IoT é retratada por [21] como a primeira evolução real da internet. Ele argumenta que a IoT adicionou à internet a sensibilidade na captação de informações, como temperatura, pressão, vibrações, entre outros. Além disso, ela consegue obter dados de lugares, plantas e animais, que até então eram inexplorados de forma autônoma e conectada. Para ele, esses fatores resultam no progresso da humanidade como um todo, uma vez que boa parte da evolução se deu pela capacidade de comunicação interpessoal, ou através de registros escritos. Com a IoT essa capacidade de comunicação é incorporada aos dispositivos para que eles consigam interagir entre si.

Considerando o trabalho de [5], o real significado e funcionalidade da IoT podem ser melhor compreendidos a partir da sua divisão em 6 componentes principais: identificação, detecção, comunicação, computação, serviços e semântica. Os objetos na rede devem ser capazes de ser identificados para que haja troca de informação. Devem haver formas de detecção dos dados, para que possam ser processados ou armazenados. Os elementos heterogêneos podem se comunicar utilizando diferentes tecnologias de comunicação para transmitir os dados, como *WiFi*, *Bluetooth*, RFID ou NFC. Unidades de processamento ou aplicações são incluídas para prover capacidade computacional à IoT. Os serviços podem ser relacionados à identidade dos objetos, à agregação de informações, para colaboração entre si ou serviços ubíquos. A semântica representa a habilidade de extrair conhecimento a partir do processamento dos dados para provisão dos serviços desejados. [5] também define alguns desafios que devem ser superados em IoT, os quais

são detalhados a seguir:

- **Disponibilidade:** representa a habilidade dos serviços estarem disponíveis a qualquer momento para usuários que podem estar em locais diferentes.
- **Confiabilidade:** quanto maior for a taxa de serviços entregues, maior a confiabilidade do sistema. A disponibilidade de sistemas é garantida por meio da confiabilidade que ele oferece.
- **Mobilidade:** a movimentação dos usuários e sensores devem ser considerada na provisão dos serviços IoT. O serviço deve permanecer disponível quando o elemento está em movimento ou quando ele é transferido para outro *gateway*.
- **Desempenho:** a dificuldade na análise do desempenho de serviços IoT se encontra na dependência da avaliação de diversos subcomponentes com tecnologias diferentes.
- **Escalabilidade:** ao incluir novos dispositivos, serviços ou funcionalidades, a qualidade do serviço oferecido deve ser mantida.
- **Interoperabilidade:** a variedade de fabricantes e especificações devem ser superados para que elementos heterogêneos nas rede IoT se comuniquem entre si.
- **Segurança:** a extensa quantidade de informações que trafega na internet, as vulnerabilidades de dispositivos e a ausência de um padrão comum e de uma arquitetura específica para a segurança em IoT faz com que a segurança seja um dos grandes desafios dessas redes. Essa questão será melhor abordada na Seção 2.2.
- **Gerenciamento:** a grande quantidade de dispositivo IoT conectados requer um controle e gerenciamento eficiente de falhas, configuração, tarifação, performance e segurança.

A estrutura de IoT baseada em três camadas (aplicação, percepção e rede) é adotada por vários trabalhos, como [2], [42], [95] e [86]. A Figura 2.1 mostra a estrutura de IoT com três camadas e a seguir há uma breve descrição delas:

- **Percepção:** Essa camada é a porta de entrada dos dados para a rede IoT. Por ela os dados do ambiente são coletados e convertidos em dados que poderão ser processados na camada de rede.
- **Rede:** É responsável pelo processamento dos dados, e faz o gerenciamento da transmissão das informações. Interliga as camada de aplicação e percepção.
- **Aplicação:** A camada de aplicação utiliza os dados processados na camada de rede para alguma finalidade. Exemplos são: interfaces de atuação nos dispositivos e programas gerenciadores dos dados.

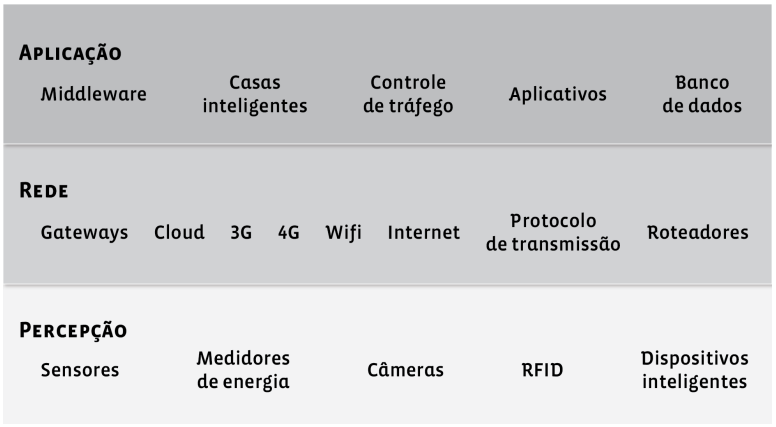


Figura 2.1: Estrutura de IoT baseada em três camadas.

2.2 Segurança

À medida que o número de dispositivos conectados aumenta, a preocupação com a segurança também cresce. Assim, para prevenir que problemas relacionados à privacidade e segurança ocorram, as questões referentes à proteção do sistema contra ataques devem ser abordadas desde o início do planejamento do projeto. Tais questões não podem ser tratadas como um complemento que será incluído em versões futuras do sistema. Com isso, é necessário discorrer sobre os princípios que devem ser cumpridos para alcançar segurança na comunicação entre pessoas, softwares, processos e coisas. Para conseguir atingir esse objetivo, [45] considera que oito fatores devem ser satisfeitos:

1. **Confidencialidade:** Os dados só devem estar disponíveis para usuários autorizados.
2. **Integridade:** Deve ser garantido que os dados transferidos não sejam adulterados.
3. **Disponibilidade:** Dispositivos, serviços e dados devem estar disponíveis no momento que os usuários precisarem.
4. **Autenticação:** Objetos na rede devem ser capazes de identificar outros objetos.
5. **Soluções leves:** O poder computacional dos dispositivos deve ser levado em consideração para provisão de soluções eficientes.
6. **Heterogeneidade:** Dispositivos com capacidade, complexidade e fabricantes diferentes devem interagir entre si.
7. **Políticas:** Padrões e políticas devem garantir o gerenciamento, proteção e transmissão dos dados de forma eficiente.
8. **Sistema de gerenciamento de chaves:** O gerenciamento de chaves deve considerar o baixo poder de processamento dos dispositivos para garantir a confidencialidade dos dados através da troca de chaves.

A IoT tem como característica utilizar o poder de processamento e comunicação de dispositivos com o intuito de captar e transmitir informações que serão aplicadas a um contexto específico. Assim, cada um dos subsistemas (dispositivos, rede de comunicação, centrais de armazenamento de dados, entre outros), que se unem para formar a IoT, pode possuir vulnerabilidades. Invasores utilizam diversos tipos de ataques para se aproveitarem dessas fragilidades. Os possíveis ataques em IoT foram listados de forma estruturada por [53]. A seguir uma breve descrição de cada tipo de ataque:

1. **Falsificar, alterar, reenviar informação roteada:** Ocorre através da falsificação, alteração e reenvio dos dados. O ataque é direcionado a rota onde ocorre a troca de informação entre dois nós.

2. **Ataque Sybil:** Representa um único nó que possui várias identidades.

3. **Denial of Service (DoS):** Causa a degradação da capacidade da rede, impedindo que serviços lícitos consigam receber os recursos desejados.

4. **Ataques baseados na propriedade do dispositivo:** São ataques que podem causar impactos diferentes à rede IoT dependendo do tipo do dispositivo. É dividido em duas classes, a primeira representa ataques por meio de dispositivos de baixo nível, no caso de baixo poder de processamento, como *smartwatches*. A segunda classe utiliza dispositivos de alto nível, que são aqueles que possuem maior capacidade de processamento, como um notebook.

5. **Ataques baseados no nível de acesso:** Afeta a disponibilidade do sistema e podem ser passivos ou ativos. O ataque passivo envolve apenas o monitoramento da informação, já o ataque ativo tenta quebrar a proteção da informação e alterar algum dado importante naquela comunicação.

6. **Ataques baseados na localização do adversário:** Se baseiam no local a partir de onde o intruso está cometendo o ataque. É dividido em dois tipos: ataque interno e ataque externo. No ataque interno o atacante está dentro da rede de segurança e executa o código malicioso no próprio dispositivo de IoT. Já no ataque externo, mesmo estando fora da rede IoT, o hacker consegue acessar os dispositivos remotamente.

7. **Ataques baseados em estratégias de ataque:** O invasor tenta executar seu código malicioso no próprio dispositivo de IoT, podendo adotar duas estratégias: física e lógica. Na estratégia física, o comportamento ou estrutura dos dispositivos inserido na rede IoT deve ser alterado. Na estratégia lógica, o ataque causa mau funcionamento no canal de comunicação, sem alterações físicas dos dispositivos.

8. **Ataques baseados no nível de dano da informação:** Causa alteração nas informações monitoradas pelos sensores. São divididas em seis categorias: Interrupção, espionagem, alteração, fabricação, reenvio de mensagem (*do inglês Replay*) e homem-no-meio (*do inglês Man-in-the-middle*).

9. **Ataques baseados no hospedeiro:** Os dispositivos de IoT podem ser atacados através dos hospedeiros do sistema, podendo ser comprometidos pelo usuário, pelo software ou pelo hardware.

10. **Ataques baseados no protocolo:** O atacante pode comprometer o protocolo utilizado na comunicação do sistema de IoT, conseguindo causar desvio ou interrupção do protocolo.

11. **Ataques baseados na pilha do protocolo de comunicação:** O ataque se concentra em um dos níveis da pilha do protocolo de comunicação. Ele pode ocorrer na camada física, no *link* de dados, na rede, na camada de transporte ou na aplicação.

Uma forma de reduzir a possibilidade de ataques seria inserindo a segurança nos dispositivos antes deles serem disponibilizados comercialmente. A solução dos problemas após os usuários adquirirem o produto é mais complexa, pois muitas pessoas não se preocupam com atualizações, nem com o gerenciamento da segurança.

2.3 **Dispositivos embarcados**

Dispositivos embarcados são equipamentos produzidos para atender um propósito específico. Dentre os diversos dispositivos embarcados existentes é possível citar como exemplos os marca-passos, empregues no controle de arritmias cardíacas, e os leitores biométricos, utilizados para identificar digitais de usuários. Nesta seção serão detalhados os dispositivos embarcados mais relevantes para este trabalho. A Subseção 2.3.1 discorre sobre os cartões inteligentes. Em seguida, leitores de cartões inteligentes são mencionados na Subseção 2.3.2. A Subseção 2.3.3 aborda os FPGAs. Enfim, a Subseção 2.3.4 descreve servidores de elementos seguros.

2.3.1 Cartões inteligentes

Cartões Inteligentes (*do inglês Smart Cards - SCs*) são circuitos integrados embarcados em um cartão, como exemplos temos os cartões de crédito e os SIM *cards* de celulares. Eles são considerados resistentes a ataques físicos, uma vez que para serem violados, um hacker levaria um longo tempo tentando decriptar os protocolos criptográficos. Também seria necessário usar equipamentos técnicos específicos, como microscópio, cortador a laser, feixes de íons focalizados, computadores potentes, entre outros [59]. Devido a isso, uma de suas principais vantagens se encontra na proteção contra acessos não autorizados aos dados armazenados. Alguns cartões permitem o processamento de algoritmos criptográficos, o que traz uma maior confiabilidade na

manipulação de tais cartões. O seu tamanho conveniente permite que os usuários carreguem eles consigo a todo momento, sem maiores preocupações com roubo de dados. Algumas das diversas aplicações que podem ser citadas para os SCs são: sistemas de pagamento, comunicação móvel, identificação pessoal, autenticação, assinatura digital e criptografia.

É conveniente utilizar os SCs para gerar chaves privadas que estarão fora do alcance de invasores, pois permanecerão seguras dentro deles. Isso previne que as chaves sejam copiadas, devido à agregação de segurança à nível de hardware, que é mais confiável que utilizar exclusivamente software [28], pois os softwares são mais propensos a possuir vulnerabilidades no Sistema Operacional (SO), na comunicação com a rede ou em atualizações de programas. Assim, os SCs são eficazes na prevenção de ataques que se baseiem na integridade, autenticidade ou confidencialidade dos dados, como o *Man-in-the-middle* ou *Replay*.

SCs possuem processador, ROM, memória não volátil, e RAM, alguns também incluem um co-processador criptográfico. Podem possuir contato elétrico ou ser sem fio, estes últimos não serão considerados no presente trabalho. O tamanho dos SCs é definido pelo protocolo [31]. O modelo ID-1 é referente ao molde no formato de cartões de crédito, já o modelo ID-000 são as dimensões dos SIM Cards de celulares.

2.3.2 Leitores de cartões inteligentes

Os leitores de cartões inteligentes, ou leitores de SCs, são dispositivos que oferecem comunicação entre computador e SC, provendo o fluxo de dados, a alimentação elétrica e o *clock* para o chip. Eles usualmente consistem de uma memória associada, um gerador de *clock* e um microcontrolador. Em geral, são programados utilizando linguagens de programação de alto nível, como C, C++ e Java, como mostra [59].

Recai sobre estes dispositivos a responsabilidade de detectar e corrigir erros a nível de transmissão de dados pelas conexões elétricas. Também, devem evitar danos ao chip, realizando as sequências de ativação e desativação. Adicionalmente, eles fazem a conversão dos níveis de tensão aceitos pelos SCs, para a tensão do controlador associado. Todas essas funcionalidades são padronizadas pelo protocolo ISO/IEC 7816-3 [32], que será explicado na Seção 2.4.1.

Os leitores de SCs mais comumente utilizados pelos usuários são os baseados em microcontroladores, e que possuem comunicação por Barramento Serial Universal (*do inglês Universal Serial Bus - USB*). A maioria das pessoas que usa este tipo de leitor o aplica para leituras e escritas esporádicas, não havendo um uso intenso. Além disso, é comum utilizar o mesmo leitor para vários SCs, revezando um SC por vez. O acesso a leitores USB é feito por meio do protocolo CCID, o qual será melhor demonstrado na Seção 2.4.2.

2.3.3 FPGA

Arranjo de Portas Programáveis em Campo (*do inglês Field Programmable Gate Array - FPGA*) é uma tecnologia de circuitos integrados, que possui blocos lógicos com conexões

reconfiguráveis após a fabricação [48]. Foi introduzido pela empresa Xilinx em 1984, trazendo uma alternativa aos Circuitos Integrados de Aplicação Específica (*do inglês Application-Specific Integratted Circuits - ASICs*) que eram produzidos sob encomenda, e como o nome deixa claro, são feitos para propósitos específicos. Dessa forma, a produção de um FPGA resulta num produto que pode ser aplicado em diversos projetos e reutilizado da maneira que for necessária, reduzindo os custos do fabricante com Engenharia Não Recorrente (*do inglês Non-Recurring Engineering - NRE*). Contudo, o valor por unidade é maior quando comparado aos ASICs.

A facilidade de mudanças no projeto implementado, aliado ao baixo custo de design e ao rápido *time-to-market*, tornam viável para projetistas de pequena escala desenvolverem suas plataformas, evitando os altos custos associados à produção dos ASICs, como mostra [74]. Além disso, os FPGAs podem ser usados na fase de prototipagem para uma posterior produção em massa.

As vantagens que eles oferecem em relação aos microcontroladores são: possuir muito mais interfaces de I/O e poder hospedar e executar muitos processos em paralelo. Uma única implementação pode ter várias réplicas no mesmo FPGA. A utilização deles facilita a evolução do projeto, devido a capacidade de reconfiguração.

A Figura 2.2, retirada de [74], mostra que os FPGAs da Xilinx, ao longo dos anos, aumentam cada vez mais sua capacidade de armazenamento e velocidade, enquanto o preço e o consumo de potência reduzem consideravelmente.

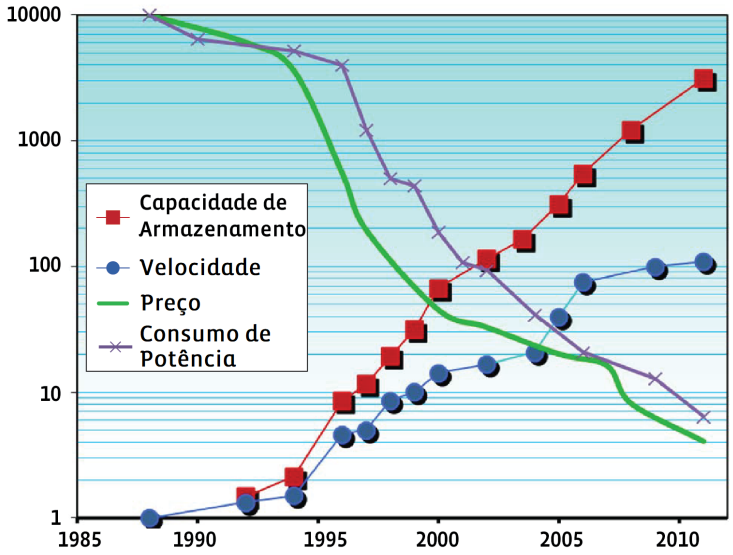


Figura 2.2: Evolução da capacidade de armazenamento, velocidade, preço e consumo de potência dos FPGAs da Xilinx, ao longo dos anos. Imagem retirada e traduzida de [74].

Sistemas baseados em FPGA podem ser implementados utilizando Linguagem de Descrição de Hardware (*do inglês Hardware Description Language - HDL*), como VHDL, System Verilog e Verilog. Apesar de não haver um consenso relacionado às etapas de desenvolvimento para FPGA, é comum utilizar o fluxo descrito na Figura 2.3 e detalhado a seguir:

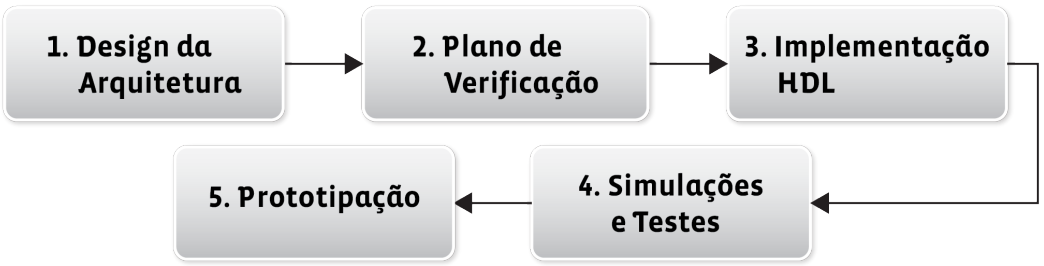


Figura 2.3: Etapas de desenvolvimento de projeto em FPGA.

1. **Design da arquitetura:** Primeiro é feita a especificação da arquitetura, consistindo de vários módulos e máquinas de estados.

2. **Plano de verificação:** Formalmente é descrita a forma como serão procedidos os testes e os critérios de validação para cada módulo.

3. **Implementação HDL:** Os módulos são implementados em linguagem de hardware para que possam ser compilados a nível de portas lógicas.

4. **Simulações e testes:** Os testes são feitos de acordo com o plano de verificação. São avaliados os tempos de processamento. Quando um erro é encontrado, deve ser avaliada em qual etapa ele foi inserido e voltar para essa etapa.

5. **Prototipação:** O código é carregado diretamente no FPGA, então ele pode ser executado.

2.3.4 Servidor de Elementos Seguros

Como é dito por [77], um Servidor de Elementos Seguros (SES) tem o objetivo de gerenciar os elementos que oferecem segurança a nível de hardware, para disponibilizar serviços de segurança, como armazenamento seguro e procedimentos criptográficos. Assim, esse dispositivo funciona por meio de uma arquitetura cliente-servidor, recebendo requisições de acesso aos elementos seguros, e retornando as devidas respostas. No caso, os elementos seguros utilizados por [77] são os SCs.

O SIM Array da empresa Implementa [30] é o exemplo de um equipamento comercial empregado para formar a estrutura de servidores de elementos seguros. Esse equipamento armazena múltiplos SCs, podendo ser adquiridos *racks* com capacidade para um, sete ou doze SIM *Boards*. Um SIM *Board* é uma placa com a capacidade para 32 cartões SIM (ISO/IEC 7810 ID-000 [31]). O processador dele é um Intel Celeron M com um núcleo de 1GHz. A

comunicação é feita por meio de conexão TCP/IP. A troca de mensagens para configuração, manutenção do sistema e interface com os SCs é realizada por meio do protocolo telnet.

O gerenciamento de múltiplos SIM *cards* e a comunicação com a rede são as maiores vantagens desse sistema. Mas o sistema possui limitações, por não permitir comunicação direta com o computador, por meio de uma porta USB, por exemplo. Isto obriga que o usuário esteja conectado a rede para acessá-lo, e inclui atrasos no tempo de resposta, que são inerentes a esse tipo de comunicação. Somado a isso, seria interessante que ele permitisse transmissões por meio de protocolos de comunicação mais seguros, não se restringindo ao telnet. Ser limitado à utilizar apenas cartões SIM, é outra desvantagem a ser considerada.

2.4 **Protocolos**

Nesta seção são apresentados os protocolos utilizados no desenvolvimento do sistema proposto pelo presente trabalho. A Subseção 2.4.1 descreve o protocolo ISO/IEC 7816-3 [32]. O CCID é detalhado na Subseção 2.4.2. Por último, a especificação PC/SC é evidenciada na Subseção 2.4.3.

2.4.1 ISO/IEC 7816-3

O ISO/IEC 7816-3 [32] é o protocolo que constitui as características de alimentação elétrica, estrutura de sinais e troca de informações entre leitores e SCs. Isso permite que os SCs e os leitores possam interagir entre si, de forma padronizada.

Esse protocolo define três classes para as condições operacionais dos SCs, relacionando o nível de tolerância para a tensão de alimentação do SC e a corrente. A descrição de cada classe está na Tabela 2.1.

Tabela 2.1: Descrição das classes de condições operacionais para SCs, de acordo com o padrão ISO/IEC 7816-3.

Classe	Tensão	Tolerância	Corrente
A	5V	4,5V - 5,5V	60mA
B	3V	2,7V - 3,3V	50mA
C	1,8V	1,62V - 1,98V	30mA

Uma aplicação em um computador hospedeiro, que se comunica com SCs para realização de alguma tarefa, utiliza Unidades de Dados do Protocolo de Aplicação (*do inglês Application Protocol Data Units - APDUs*) como contêiner para fazer a transmissão dos comandos. Todo o processo de comunicação, e a estrutura própria das APDUs estão descritos no protocolo [33]. Os leitores transmitem as APDUs para os SCs através da interface serial, elas são encapsuladas em Unidades de Dados do Protocolo de Transporte (*do inglês Transport Protocol Data Units - TPDUs*). O ISO/IEC 7816-3 [32] define dois protocolos de transmissão, o T=0 e o T=1, que

são detalhados mais a frente, nesta mesma seção. Contudo, exitem protocolos de transmissão que são estabelecidos para finalidades específicas, como mostra [59], o protocolo de transmissão T=14 foi utilizado exclusivamente na Alemanha para cartões de telefones públicos.

Os parâmetros iniciais enviados pelo SC, para configuração do leitor, são conhecidos como ATR. A sua estrutura será melhor abordada a seguir.

2.4.1.1 *ATR*

Os primeiros bytes enviado por cada SC, após receber o sinal de *reset*, são chamados de Resposta ao *Reset* (*do inglês Answer To Reset - ATR*). Para qualquer SC a transmissão de tais bytes é feita a uma frequência máxima de 5Mhz, para garantir que qualquer leitor que esteja de acordo com o padrão ISO/IEC 7816-3 [32], possa receber esses parâmetros iniciais, mesmo que ele não dê suporte ao protocolo de transmissão do SC. O tamanho máximo de um ATR é de 33 bytes. Ele carrega parâmetros para configuração do leitor, que estão relacionados ao protocolo de transmissão e preferências de *clock* e tensão do SC. A estrutura básica de um ATR é mostrada na Figura 2.4, retirada de [59]. A descrição dos parâmetros incluídos no ATR é feita a seguir:

1. **Caractere inicial (TS):** é o primeiro byte a ser enviado. Descreve a convenção de bits utilizada pelo SC, e só possui dois valores possíveis: ’3B’ e ’3F’. Quando ele é ’3B’ chama-se de convenção direta, nela o nível de tensão alto representa 1 e o nível baixo 0. Se for recebido ’3F’ temos a conversão inversa, nela o 1 é o nível de tensão baixo e o 0 é o nível alto.

2. **Caractere de formato (T0):** indica quais caracteres de interface serão enviados a seguir, e a quantidade de bytes de histórico que o ATR possui.

3. **Caracteres de interface (TAi, TBi, TCi e TDi):** eles definem os parâmetros para o protocolo de transmissão. Como valores padrão já são definidos, esses bytes podem ser omitidos se o SC não desejar alterações no processo de transmissão predefinido. Esses caracteres carregam o protocolo de transmissão a ser utilizado, a frequência máxima aceita pelo SC, as classes de tensão aceitas, o tempo de espera entre bytes, o tempo de espera entre blocos de bytes, entre outros.

4. **Caracteres de histórico (T1,T2,...TK):** eles podem ser omitidos, mas, em geral são usados para identificar o sistema operacional do SC.

5. **Caracteres de Checagem (TCK):** eles mantêm o somatório de checagem dos bytes, começando do T0, até o último byte antes do TCK. São utilizados para verificar se o ATR foi transmitido corretamente. O TCK não é utilizado quando o SC suporta apenas o protocolo de transmissão T=0.

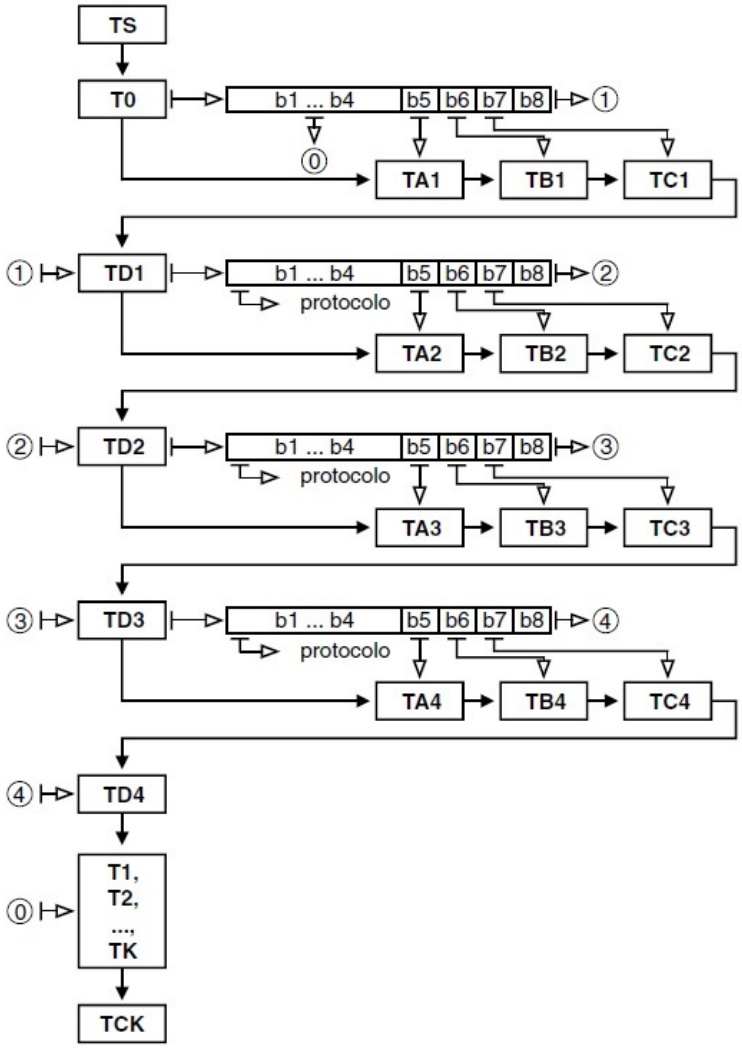


Figura 2.4: Estrutura básica do ATR. Imagem retirada e traduzida de [59].

2.4.1.2 *Protocolo de transmissão T=0*

A transmissão por meio do protocolo T=0 é orientada a bytes, *half-duplex* e assíncrona. Ele foi pensado para ser simples e utilizar pouca memória. Como um caractere (8 bits) é transmitido individualmente, o protocolo possui um bit de paridade para detecção de erros em cada byte, isso permite a retransmissão dele em caso de falhas. O comando desse protocolo é estruturado com 5 bytes de cabeçalho seguido pelos bytes de dados, como mostra a Tabela 2.2. O CLA é o byte que define a classe do comando, o INS é a instrução a ser executada pelo SC, o P1 e o P2 são parâmetros específicos de cada instrução e o P3 estabelece o número de bytes de dados que estarão em DATA FIELD.

Tabela 2.2: Descrição da mensagem no protocolo T=0.

CLA	INS	P1	P2	P3	DATA FIELD
1 byte	1 byte	1 byte	1 byte	1 byte	0 a 255 byte

Pela Figura 2.5 é possível ver que a transmissão começa com o leitor enviando o

cabeçalho. Após o SC receber essa mensagem, ele retorna um byte de confirmação, chamado de ACK. Se este byte retornado for igual ao valor do byte INS no cabeçalho, significa que a transmissão foi bem sucedida e que os bytes de dados podem ser enviados pelo leitor. Porém, se o ACK for o resultado de um ou-exclusivo do INS com o valor 'FF', apenas um byte deve ser enviado, então o leitor deve esperar outro ACK para poder enviar os bytes restantes. Após a completa transmissão dos bytes de dados, o SC encaminha dois bytes, o SW1 e o SW2, que informam seu status final e se ele possui dados para enviar, respectivamente.

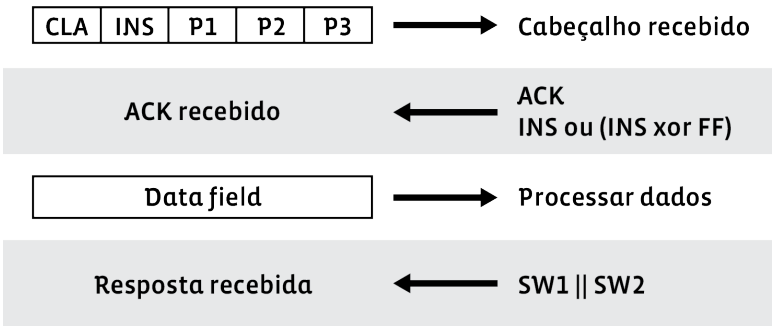


Figura 2.5: Processo de troca de mensagens do protocolo T=0.

Após o processo descrito no parágrafo anterior, o leitor pode enviar uma instrução de requisição de resposta (GET RESPONSE), a qual solicita ao SC que envie a quantidade de dados relatada em SW2, como resposta ao primeiro comando enviado pelo leitor.

2.4.1.3 *Protocolo de transmissão T=1*

O protocolo de transmissão T=1, também é *half-duplex* e assíncrono, contudo os bytes são enviados como uma cadeia de caracteres, agrupados em blocos. Como um bloco é a menor unidade de dados transmitida, a detecção de erros é feita sobre a cadeia inteira de bytes, fazendo com que, se ao menos um byte esteja incorreto, todo bloco seja reenviado. Assim, os últimos bytes do bloco transmitem o código de detecção de erro, que pode ser por Verificação Longitudinal de Redundância (*do inglês Longitudinal Redundancy Check - LRC*), ou o pela Checagem Cíclica de Redundância (*do inglês Cyclic Redundancy Check - CRC*). A Tabela 2.3 descreve brevemente ambos os métodos. A checagem do bit de paridade do caractere também é feita, como forma complementar a verificação por blocos.

Tabela 2.3: Descrição dos métodos de detecção de erro utilizados no protocolo T=1.

Método	Bytes	Descrição
CRC	2	Baseado no cálculo do resto da divisão de polinômios
LRC	1	Calculado através do ou-exclusivos de todos os bytes do bloco

Para que não haja retransmissão ou sinalização de erros, o envio de caracteres ou blocos deve respeitar os tempos de guarda e os tempos de espera. O tempo de guarda é o período mínimo

que deve ser cumprido entre envios de bytes ou blocos. O tempo de espera é o período máximo que pode ser aguardado, antes de alguma falha na transmissão ser sinalizada. A descrição dos tempos que devem ser respeitados pode ser observada a seguir:

- **CGT:** Tempo de Guarda entre Caracteres (*do inglês Character Guard Time - CGT*). É o intervalo de tempo mínimo que deve ser respeitado ao enviar dois caracteres consecutivos.
- **BGT:** Tempo de Guarda entre Blocos (*do inglês Block Guard Time - BGT*). É o intervalo de tempo mínimo que deve haver entre o último byte enviado e o primeiro byte recebido como resposta.
- **CWT:** Tempo de Espera entre Caracteres (*do inglês Character Wait Time - CWT*). É a duração limite para que dois caracteres consecutivos no mesmo bloco sejam enviado.
- **BWT:** Tempo de Espera entre Blocos (*do inglês Block Wait Time - BWT*). É o prazo máximo para que o SC comece a enviar o bloco de resposta, após o leitor ter transmitido completamente a requisição para o SC.

Como mostrado na Tabela 2.4, o bloco consiste de 3 bytes de cabeçalho, até 254 bytes de informação e 1 ou 2 bytes para checagem de erro. O NAD (*do inglês Node Address*) serve para distinguir a origem e o destino do bloco quando houverem múltiplas conexões simultâneas. O PCB (*do inglês Protocol Control Byte*) informa o tipo de bloco que está sendo enviado e algumas informações de controle de transmissão. O LEN (*do inglês Length*) traz o número de bytes de informação no campo INF. O INF (*do inglês Information Field*) carrega os dados da aplicação, sua presença depende do tipo de bloco descrito no byte PCB e seu tamanho vem no byte LEN. No campo LRC ou CRC, há o código para verificação de erros, contando os bytes desde o NAD até o último byte de informação.

Tabela 2.4: Descrição da mensagem no protocolo T=1.

NAD	PCB	LEN	INF	LRC ou CRC
1 byte	1 byte	1 byte	0 a 254 bytes	1 ou 2 bytes

O cenário mais simples de transmissão com o protocolo T=1 ocorre quando apenas um bloco de informação é enviado e o SC responde comunicando que não houve erro. Porém, em ISO/IEC 7816-3 [32] pode ser encontrados outros contextos de transmissão como: solicitação de extensão de tempo de espera, ajuste do tamanho do campo de informação, encadeamento de blocos, manipulação de erros e ressincronização.

2.4.2 CCID

Um Leitor de SCs também pode ser chamado de Dispositivo de Interface para Cartões com Circuito Integrado (*do inglês Circuit(s) Cards Interface Device - CCID*). Esse nome, inclusive, define o protocolo em questão nesta seção. Tal protocolo permite que leitores possam interagir com computadores hospedeiros através de interface USB.

O CCID define quatro canais lógicos de troca de mensagens, que são chamados de pipes. Segue a descrição de cada um deles:

- **Pipe de controle:** esse pipe é utilizado para controlar o dispositivo USB, fazendo requisições padrão e atribuindo algumas configurações.
- **Pipe de interrupção:** é empregado no tratamento de eventos assíncronos, como detecção de problemas de hardware e inserções e remoções de SCs.
- **Pipe de bulk-out:** os comandos são enviados por esse pipe. Todo comando enviado deve receber ao menos uma mensagem de resposta.
- **Pipe de bulk-in:** é por onde as respostas aos comandos enviados são entregues.

Há três níveis diferentes de troca de mensagens que podem ser feitas por meio desse protocolo, eles são relatados abaixo:

1. **Nível de troca por TPDU:** o leitor recebe do computador hospedeiro o TPDU, T=0 ou T=1, já montado para transmitir para o SC.
2. **Nível de troca por APDU:** o hospedeiro remete APDUs para o leitor, ficando sobre a responsabilidade deste fazer a conversão para TPDUs e em seguida repassar para o SC. O leitor também converte os TPDUs recebidos do SC em APDUs e retorna para o hospedeiro.
3. **Nível de troca por caractere:** a troca de mensagem entre o leitor e o computador hospedeiro é feita byte a byte, de acordo com a quantidade de bytes que o comando diz possuir.

As mensagens do CCID são enviadas com um cabeçalho de 10 bytes, seguido pelos bytes de dados. A Tabela 2.5 mostra a estrutura do cabeçalho, nele o TIPO diz o comando que deverá ser processado. Os quatro bytes seguintes do campo COMPRIMENTO carregam a quantidade de bytes que virá após o cabeçalho. Em seguida, o byte chamado SLOT expressa para qual *slot* de SC a mensagem deve ser enviada, serve para leitores que aceitem mais de um SC. O byte que vem em seguida é o SEQ, ele guarda um número de sequência que relaciona o comando enviado com a resposta, devendo ser igual em ambos, para o hospedeiro saber que aquela resposta é referente ao comando enviado. Os três últimos bytes, intitulados RFU, são reservados para uso futuro.

Tabela 2.5: Estrutura do cabeçalho de uma mensagem em uma transmissão com o protocolo CCID.

TIPO	COMPRIMENTO	SLOT	SEQ	RFU
1 byte	4 bytes	1 byte	1 byte	3 bytes

2.4.3 PC/SC

Para fazer com que leitores, SCs e computadores de marcas diferentes pudessem se comunicar entre si, o PC/SC Workgroup desenvolveu o protocolo Computador Pessoal/Cartão Inteligente (*do inglês Personal Computer/Smart Card - PC/SC*) que pode ser encontrado em [58]. Essa especificação visa permitir que múltiplas aplicações de alto nível se comuniquem com leitores de SCs, para isso ele define uma interface de baixo nível com esses dispositivos, uma Interface de Programação de Aplicações (*do inglês Application Programming Interface - API*) que é independente do dispositivo e se comunica com aplicações de alto nível.

A Figura 2.6, retirada de [58], representa a arquitetura e os componentes desse protocolo. Segue uma breve descrição dos principais módulos:

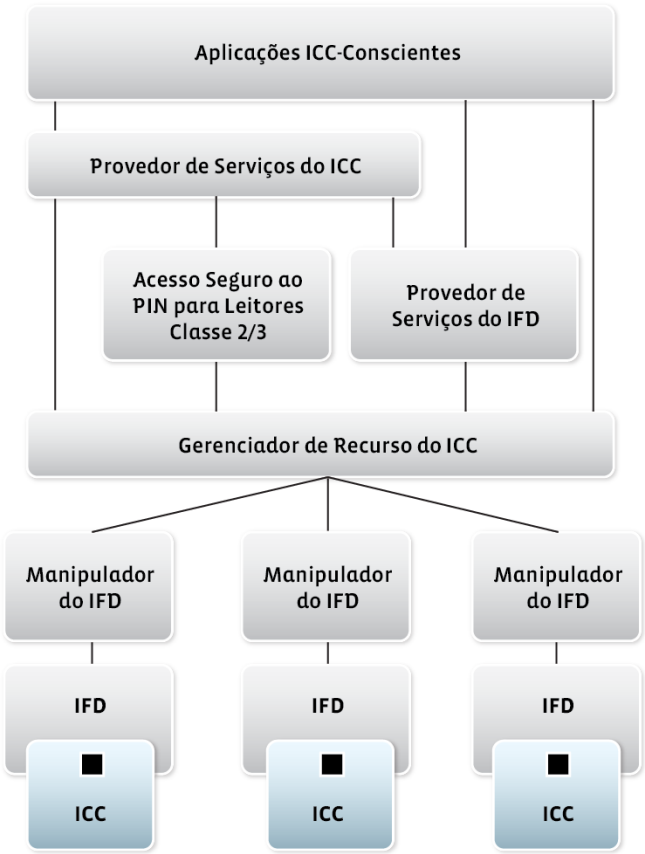


Figura 2.6: Arquitetura do protocolo PC/SC. Imagem modificada de [58].

- **ICC:** Cartão com Circuito Integrado (*do inglês Integrated Circuit Card - ICC*). Representa o SC.

- **IFD:** Dispositivo de Interface (*do inglês Interface Device - IFD*). Corresponde ao leitor de SCs.
- **Manipulador do IFD:** é o *driver* de comunicação com o computador hospedeiro.
- **Gerenciador de Recursos do ICC:** representa o gerenciador de recursos. Ele possui a informação dos leitores de SCs que estão disponíveis e quando são inseridos ou removidos. Além disso, ele controla a alocação de recursos para cada leitor e faz o gerenciamento das *threads*.
- **Provedor de Serviços do IFD:** este módulo faz a interface quando o IFD possui alguma funcionalidade estendida.
- **Provedor de Serviços do ICC:** faz o encapsulamento das funcionalidades disponibilizadas pelo cartão e deixa acessível para a API de alto nível.
- **Acesso Seguro ao PIN para Leitores Classe 2/3:** gerencia leitores que permitem acesso por meio de código de segurança (PIN).
- **Aplicações ICC-Conscientes:** é uma aplicação que mapeia as requisições de serviços do hospedeiro para requisições do SC.

3

ESTADO DA ARTE

Os trabalhos relacionados a plataforma proposta e a segurança em IoT, são analisados neste capítulo. Na Seção 3.1, é feito um mapeamento sistemático dos trabalhos que aplicam alguma técnica de segurança em IoT. Os trabalhos que implementam leitores de SC baseados em FPGA são apresentados na Seção 3.2. Já na Seção 3.3 estão trabalhos que utilizam algum dispositivo de agrupamento de SC.

3.1 Segurança em IoT

Foi feita uma revisão sistemática da literatura relacionada a segurança em IoT, com o intuito de: avaliar os trabalhos que aplicam alguma técnica de segurança para IoT, e sinalizar as contribuições do presente trabalho dentro desse contexto.

A revisão foi feita a partir de três mecanismos de busca de artigos: IEEE, ACM e Science Direct. Uma busca inicial foi feita entre os trabalhos publicados até o mês de Maio de 2017, que tivessem no campo de palavras chave (*do inglês keywords*) ambos os seguintes termos: *IoT* e *security*. Na busca inicial foram excluídos capítulos de livros. Não foi estabelecida uma limitação mínima na data de publicação, contudo o trabalho mais antigo que atende o critério anterior foi publicado no ano de 2011. Essa primeira busca resultou em 119 trabalhos. A partir deles foi feita uma nova busca, na qual só foram incluídos os artigos que possuem alguma técnica de segurança aplicada ao contexto de IoT. Foram excluídos pôsteres, trabalhos que não mostram claramente a utilização da segurança e trabalhos que não se enquadram em IoT. O refinamento da primeira busca resultou em 45 artigos.

Por meio da análise dos 45 artigos, quatro categorias de classificação dos trabalhos foram definidas. As categorias podem ser visualizadas hierarquicamente pela Figura 3.1, e são descritas a seguir:

- **Camada da IoT onde a segurança é integrada:** os trabalhos são distribuídos segundo a camada da IoT onde eles incorporam a segurança. Podendo estar nas camadas de: aplicação, rede ou percepção. Elas foram descritas na Seção 2.1.

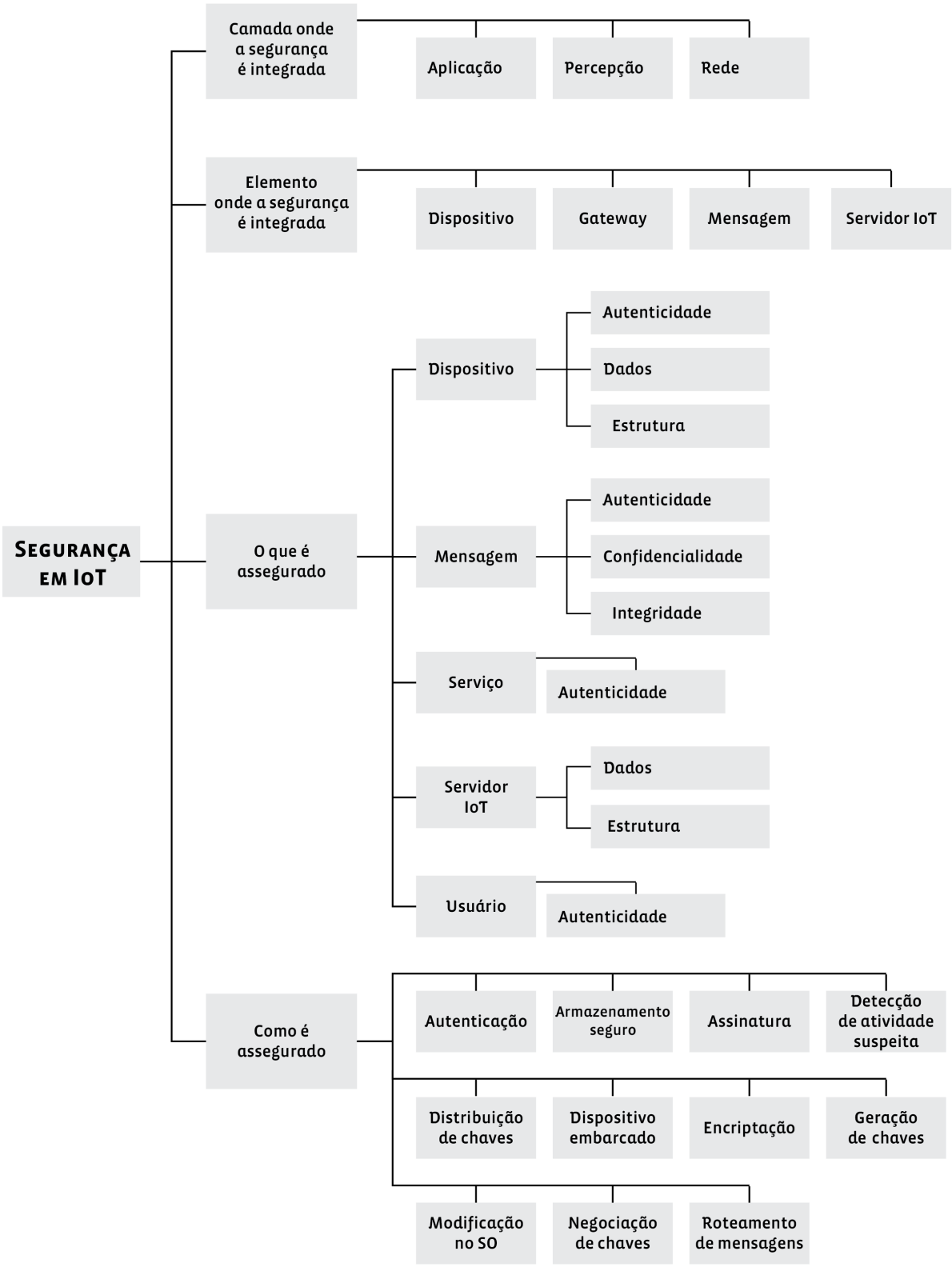


Figura 3.1: Diagrama hierárquico das categorias de classificação de trabalhos em segurança de IoT.

- **Elemento da IoT onde a segurança é integrada:** foram identificados quatro elementos onde a segurança pode ser integrada: dispositivo, gateway, mensagem e servidor IoT.
  - **Dispositivo:** são dispositivos na ponta da IoT, como sensores ou atuadores.
  - **Gateway:** pode ser um elemento de hardware, como roteadores e switches, ou software, como um programa gerenciador de recepção e transmissão de mensagens.
  - **Mensagem:** quando há proposição do protocolo de transmissão ou do formato da informação, que trafega entre quaisquer elementos na rede IoT.
  - **Servidor IoT:** serve como denominação para armazenamento (*storage*), middlewares, computação na nuvem e processamento de dados.
- **O que é assegurado:** o dispositivo onde a segurança é implementada, nem sempre é o componente da infraestrutura de IoT que será, de fato, assegurado. Os seguintes elementos foram assegurados por ao menos um trabalho: dispositivo, mensagem, serviço, servidor IoT e usuário.
  - **Dispositivo:** podem ser assegurados, a sua autenticidade, os dados que estão armazenados nele, ou a própria estrutura. A estrutura é assegurada no caso de ataques físicos, como análise de potência.
  - **Mensagem:** a segurança pode focar na autenticidade, na confidencialidade, ou na integridade das mensagens.
  - **Serviço:** a autenticidade de algum serviço fornecido é verificada.
  - **Servidor IoT:** assegura os dados armazenados ou a estrutura do servidor. Um exemplo, é a proteção do servidor contra ataques DDoS.
  - **Usuário:** considera a autenticidade do usuário. Assegura que apenas pessoas autorizadas utilizem as aplicações e serviços do sistema de IoT.
- **Como é assegurado:** define a estratégia utilizada pelo trabalho para alcançar a segurança. Ela pode ser obtida através de: autenticação, encriptação, assinatura, negociação de chaves, detecção de atividade suspeita, armazenamento seguro, chave baseada no hardware, roteamento de mensagens, inclusão de dispositivo embarcado, geração de chaves, modificação no SO ou distribuição de chaves.
  - **Autenticação:** utiliza alguma técnica para comprovar a autenticidade das partes comunicantes.
  - **Armazenamento seguro:** os dados são armazenados de forma segura, dificultando o roubo de dados.

- **Assinatura:** forma de garantir a integridade da mensagem, estabelecer a autenticidade dela e assegurar que o autor não possa negar sua autoria (não repúdio).
- **Detecção de atividade suspeita:** o comportamento do sistema é monitorado para verificar se há componentes da rede causando instabilidade no sistema. Esta instabilidade pode ter sido introduzida por algum invasor.
- **Dispositivo embarcado:** o trabalho inclui um dispositivo embarcado ou componente eletrônico que agrega segurança ao sistema de IoT.
- **Distribuição de chaves:** as chaves geradas são distribuídas entre os usuários, dispositivo ou serviços, de forma a não permitir que o invasor consiga obtê-las.
- **Encriptação:** os dados são disfarçados para evitar que pessoas não autorizadas observem a informação.
- **Geração de chaves:** é aplicada alguma técnica para geração de chaves criptográficas de forma segura. Pode ser gerada exclusivamente através de software, ou a partir de alguma característica do hardware.
- **Modificação no SO:** o SO utilizado nos dispositivos, no gerenciamento da rede, ou no servidor IoT, pode ser alterado para agregar alguma forma de segurança.
- **Negociação de chaves:** chaves criptográficas devem ser trocadas de forma segura. Garante que invasores não rastreiem a negociação das chaves.
- **Roteamento de mensagens:** a proteção está no processo de transmissão da mensagem, seja no protocolo utilizado, ou na otimização do roteamento através do caminho mais seguro.

A maioria dos trabalhos se concentram na camada de rede, como mostra a Figura 3.2, que possui a quantidade de trabalhos em cada uma das camadas da IoT. A Figura 3.3 traz a quantidade de trabalhos por elemento da IoT onde a segurança pode ser integrada. Os elementos mais utilizados para embarcar a segurança são os dispositivos na ponta da IoT, enquanto as mensagens são menos aproveitadas com essa finalidade. Pela Figura 3.4 é possível observar que técnicas de autenticação são as mais comuns entre os trabalhos que inserem segurança na IoT. Apenas um trabalho foca na distribuição segura de chaves criptográficas. Analisando a quantidade de trabalhos para cada elemento da IoT assegurado, que está na Figura 3.5, é constatado que a maior parte dos trabalhos trazem alguma forma de agregar segurança às mensagens transmitidas. Mais especificamente, a autenticidade das mensagens é o elemento mais utilizado pelos trabalhos para agregar segurança. Os elementos menos assegurados são a autenticidade dos serviços e a estrutura do servidor IoT, que possuem apenas um trabalho cada.

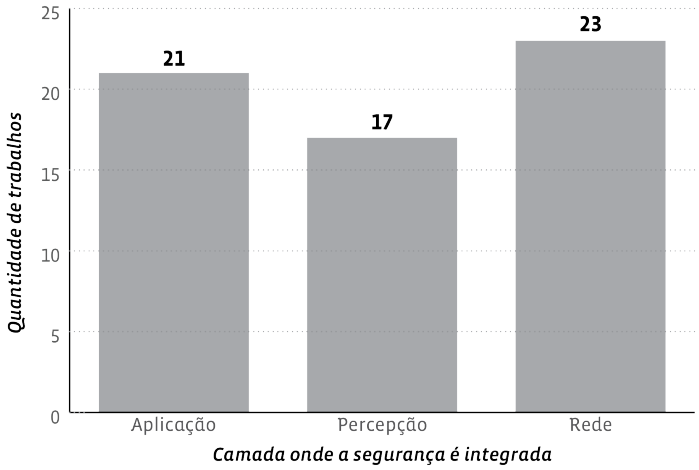


Figura 3.2: Quantidade de trabalhos classificados em cada uma das camadas da IoT.

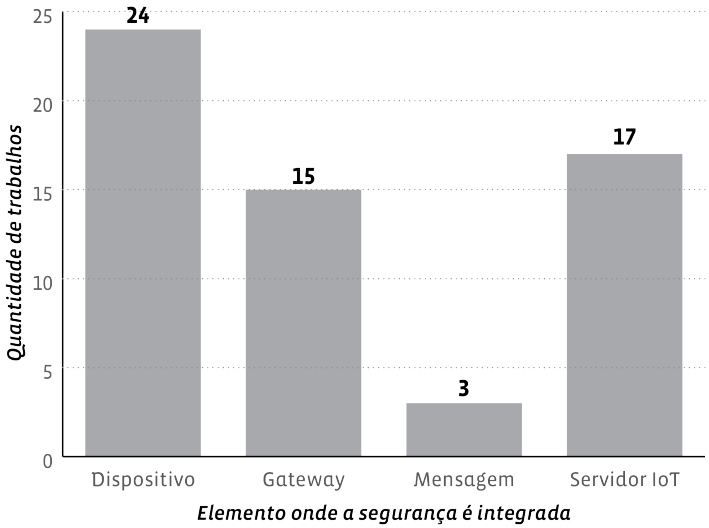


Figura 3.3: Quantidade de trabalhos para cada elemento da IoT onde a segurança pode ser integrada.

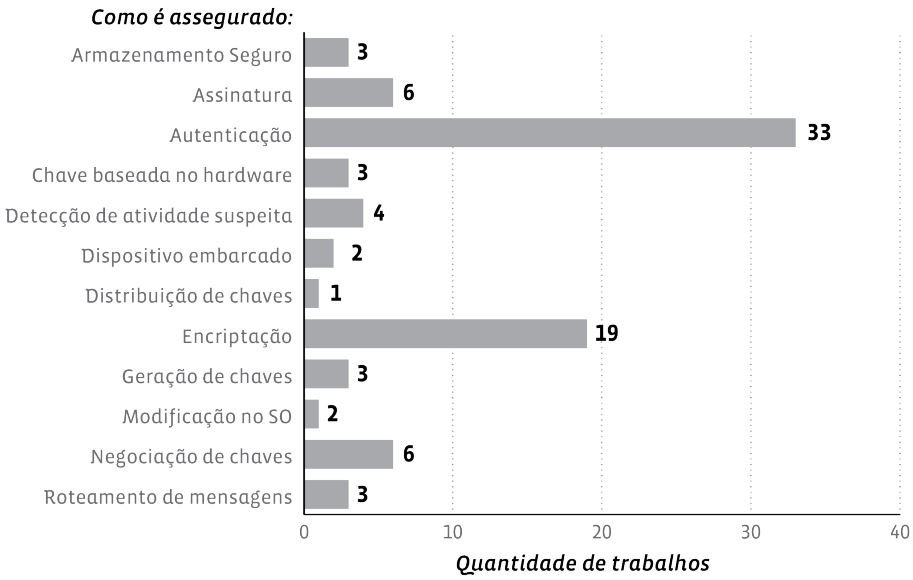


Figura 3.4: Quantidade de trabalhos para cada uma das formas de agregar segurança na IoT.

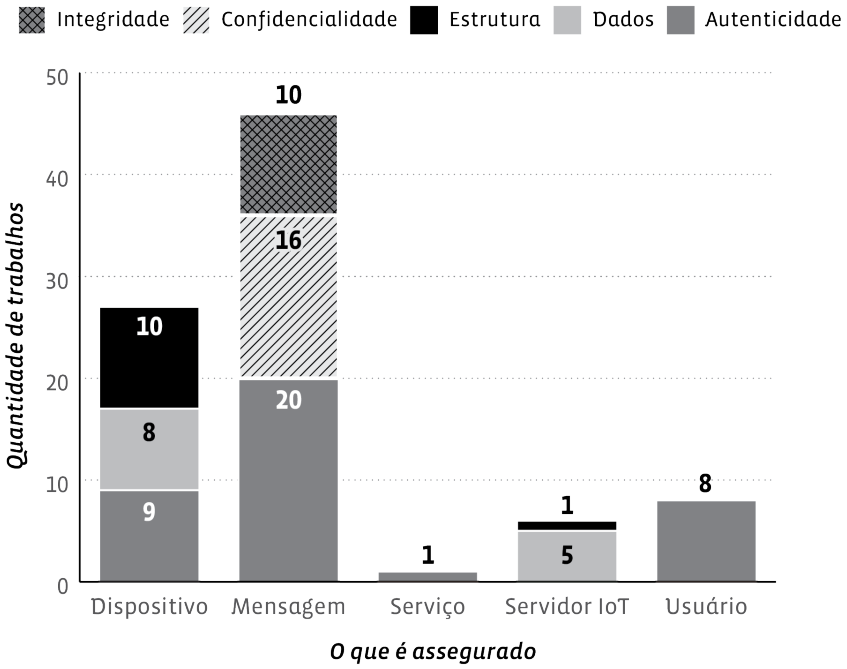


Figura 3.5: Quantidade de trabalhos para cada elemento da IoT que é assegurado.

A Tabela 3.1 descreve a classificação dos trabalhos, relacionando a camada onde ele se aplica e o elemento onde a segurança é integrada. Por ela, é possível ver que a quantidade de trabalhos representa a proximidade que o elemento tem com a camada da IoT, não desconsiderando o fato do mesmo trabalho poder assegurar mais de uma camada. No caso dos dispositivos, a maioria dos trabalhos estão na camada de percepção e a minoria na camada de aplicação, como seria esperado já que por definição os dispositivos estão na ponta da rede. Já para os *gateways*, a maioria está na camada de rede, dado que ele é um elemento dessa camada. O servidor IoT, tem mais trabalhos na camada de aplicação e em segundo lugar está a camada de rede, mostrando o comportamento integrado que ele tem de processar as mensagens, fazer o transporte e executar alguma aplicações e serviços. Considerando a integração de segurança às mensagens, não há trabalhos que as utilizem para assegurar a camada de aplicação, estando a maioria na camada de rede.

Tabela 3.1: Classificação dos trabalhos de segurança em IoT, relacionando: a camada (linhas) e em qual elemento (colunas) a segurança é integrada. Em cinza claro: a camada e em qual elemento o SCC se inclui. Em cinza escuro: a camada e em qual elemento o SCC é aplicado neste trabalho.

	Dispositivo	Gateway	Servidor IoT	Mensagem
Aplicação	[1][37] [87][88]	[1][27][37][68] [71] [81][87][96]	[1][11][13][16][27] [34][35][36][37] [40][49][68][71] [81][87][88][96]	-
Percepção	[1][10][29][37] [38][43][46] [52] [55][62][65][66] [70][72][85][87] [89][90][91][92]	[1][37] [61][87]	[1][37][87]	[10]
Rede	[1][10][20] [37][39] [51] [55][87][88]	[1][19][20] [23][27] [37][44][56][63][68] [71][81][87][96]	[1][27][34][35] [37][68][71][81] [87] [88][96]	[10][54] [75]

A Tabela 3.2 mostra a classificação dos trabalhos de acordo com, o que é assegurado e como a segurança é integrada. Nem sempre a forma como a segurança é integrada, está diretamente ligada ao elemento que é assegurado. Um exemplo é a impossibilidade de utilizar armazenamento seguro para autenticar serviços, porém seria possível haver trabalhos que fizessem ambas as coisas, mas sem relação de causa e efeito. Para todos os elementos que foram assegurados, a estratégia mais adotada é a de autenticação, principalmente para as mensagens, que também possuem muitos trabalhos focando na encriptação. É possível observar pela Tabela 3.2 as lacunas que ainda não foram exploradas no contexto de segurança em IoT. As estratégias menos apontadas são as de geração de chaves e utilização de dispositivos embarcados. Por isso, seria interessante haver trabalhos que destacassem a utilização destas estratégias. Devendo-se considerar a sua relação com elementos de extrema importância, porém pouco estudados, como a estrutura do servidor IoT, a autenticação de serviços e a autenticidade dos usuários.

Tabela 3.2: Classificação dos trabalhos de segurança em IoT, relacionando: como (linhas) e o que (colunas) é assegurado. Em cinza claro: como e o que pode ser assegurado pelo SCC . Em cinza escuro: como e o que é assegurado pelo SCC neste trabalho.

	Dispositivo			Mensagem			Serviços	Servidor IoT		Usuário
	Autent.	Dados	Estrutura	Autenticidade	Confidencialidade	Integri.	Autent.	Dados	Estrutura	Autent.
Autenticação	[35][39][43] [56][62] [63][81] [89][92]	[35] [72] [90]	[19] [72] [90]	[1][10][16][34] [40][44][51][54] [55][63][65][71] [75][87][88][90]	[1][10][16][23] [34][35][37][51] [55][63][65][71] [75][87][88][90]	[10][23][35] [37][40] [44][54] [71][81]	[35]	[13][35] [36][49] [81]	-	[13][19] [29][35] [36][38] [49][89]
Armazenam. seguro	[81]	-	-	-	-	[81]	-	[36][49] [81]	-	[36][49]
Assinatura	[81]	-	-	[10][40] [54][71]	[10][23][71]	[10][23][40] [54][71][81]	-	[81]	-	-
Chave bas. no hardware	-	[72] [85]	[27][70] [72][85]	-	-	-	-	-	-	-
Detecção de ativ. suspeita	-	-	[27]	[27][61][68]	-	-	-	-	[11]	-
Dispositivo embarcado	-	[66]	[66]	[68]	-	-	-	-	-	-
Distribuição de chaves	[43]	-	-	-	-	-	-	-	-	-
Encriptação	[35][63]	[35] [85] [90] [91]	[85] [90] [91]	[1][10][16][34] [51][55][63][65] [71][75][87] [88][90][96]	[1][10][16][23] [34][35][51][55] [63][71][75] [87][88][90]	[10][23] [35][71]	[35]	[13][35]	-	[13][35]
Geração de chaves	-	[72] [85]	[70][72] [85]	-	-	-	-	-	-	-
Mod. no SO	-	[46][52]	[46][52]	-	-	-	-	-	-	-
Negociação de chaves	[62][63] [92]	-	-	[1][16][63]	[1][16][63]	-	-	[36]	-	[36]
Rot. de mensagens	-	-	-	[27][61]	-	[20]	-	-	-	-

3.2 Leitores de SCs baseados em FPGA

Nesta seção são reunidos os trabalhos que utilizam FPGA para fazer a leitura de dados de SCs. No trabalho de [41], ele implementa um leitor de SCs em FPGA, apresenta simulações, e uma breve descrição dos recursos de FPGA utilizados. [47] também faz uma implementação de leitor de SCs em FPGA, que foi sintetizado tanto em placas de FPGA da Altera quanto da Xilinx. Ele compara com um leitor de SC desenvolvido para Dispositivos Lógicos Programáveis Complexos (*do inglês Complex Programmable Logic Devices - CPLDs*). Ambos os trabalhos anteriores são baseados na arquitetura proposta pela Xilinx em [18]. Como ela é uma arquitetura bem simplificada, não é possível fazer checagem da ocorrência de erro de transmissão através do bit de paridade, não possui tratamento do ATR e apenas suporta o próprio SC, o cartão ACOS1. Portanto, não é possível considerar tais trabalhos como uma implementação completa do protocolo ISO/IEC 7816-3 [32]. Estes trabalhos não entram nos critérios da revisão sistemática apresentada na Seção 3.1 pois não são aplicados à segurança em IoT.

Nessa dissertação é apresentada uma arquitetura de leitor de SCs baseada em FPGA que é capaz de manipular as mensagens de ATR e fazer a leitura de vários tipos de SCs, com diferentes ATRs, além de ter a checagem de erro de transmissão por meio do bit de paridade. A implementação em hardware é descrita no Capítulo 4. O desempenho do sistema é avaliado e ele é aplicado para aumentar o nível de segurança em um cenário de IoT no Capítulo 5.

A Tabela 3.3 foi montada a partir da análise e comparação das contribuições de cada trabalho citado nesta seção.

Tabela 3.3: Comparação das contribuições entre trabalhos que desenvolvem leitor de SCs baseado em FPGA.

Contribuição	Mandi	Koul	Xilinx	**Este trabalho**
Implementação em hardware	X	X	X	X
Utilização de recursos	X	X	-	X
Proposta de arquitetura	-	-	X	X
Gerenciamento de ATR	-	-	-	X
Checagem de erro	-	-	-	X
Suporte a diversos tipos de SCs	-	-	-	X
Avaliação do desempenho	-	-	-	X
Aplicação em segurança de IoT	-	-	-	X

3.3 Agrupamento de SCs

A ideia de agrupar SC para fazer o processamento paralelo de tarefas foi introduzido por [14]. No trabalho ele forma uma *Java Cards Grid* e desenvolve um *framework* que faz o

gerenciamento da computação distribuída entre cartões. A plataforma de hardware que dá suporte ao projeto foi montada com leitores de SC USB conectados à um computador hospedeiro, que faz o gerenciamento deles. Uma evolução do trabalho anterior é apresentada por [12], o qual aplica a plataforma de agrupamento de SCs a diferentes domínios com o objetivo de agregar segurança às aplicações. A utilização de Java Card Grids também é feita por [15] para oferecer serviços colaborativos em alto nível que são embarcados nos SCs e podem ser utilizados remotamente pelos clientes.

[79] utiliza SCs agrupados, chamado por ele de SC Grids, para criar um servidor de SCs, que processa o protocolo EAP-TLS. Isso permite que seja oferecida computação segura como serviço, em uma infraestrutura de computação nas nuvens. O conceito de Nuvem de Elementos Seguros (*do inglês Cloud of Secure Elements - CoSE*) introduzido por [80], tem como base o armazenamento de aplicações em elementos seguros localizados na nuvem, para isso fazem uso de Grids de Elementos Seguros (*do inglês Grid of Secure Elements - GoSE*) que compreende um hardware gerenciador de vários SCs com comunicação TCP. Em tal trabalho, um dispositivo móvel serve de *proxy* para comunicação entre um leitor de NFC e a GoSE. A arquitetura global da CoSE foi publicada em [76], mas foi implementada pela primeira em [78], onde o dispositivo utilizado para a interface com os SCs era um servidor de elementos seguros com porta TCP.

Dispositivos de agrupamento de SCs são empregados por [4] como servidores para autenticação mútua entre máquinas virtuais e o *hypervisor* Xen, garantindo a comunicação segura entre eles.

Todos esses trabalhos têm em comum a utilização de um SES, como apresentado na Seção 2.3.4. Eles trazem a desvantagem de centralizar as requisições, que podem gerar atrasos devido a alta demanda em IoT, e serem mais suscetíveis a ataques do tipo DDoS. Além disso, o alto custo pode inviabilizar a sua utilização em sistema de pequeno porte. Com o SCC, é possível integrar a plataforma diretamente aos *gateways*, de forma distribuída, não gerando os atrasos inerentes da comunicação cliente-servidor.

4

O SMART CARD CLUSTER

Este capítulo é dedicado à apresentação da plataforma proposta. A Seção 4.1 possui uma visão geral do sistema. Um caso de uso será detalhado na seção Seção 4.2. A Seção 4.3 mostrará o desenvolvimento feito em FPGA. A interface eletrônica de comunicação com os SCs é descrita na Seção 4.4. Na Seção 4.5 o conector de transmissão física é apresentado. Por fim, o *driver* de dispositivo será destrinchado na Seção 4.6.

4.1 Visão Geral

O *Smart Card Cluster* (SCC) é uma plataforma de segurança para IoT que faz agrupamento e gerenciamento de vários SCs utilizando FPGA. A sua arquitetura geral é apresentada na Figura 4.1.

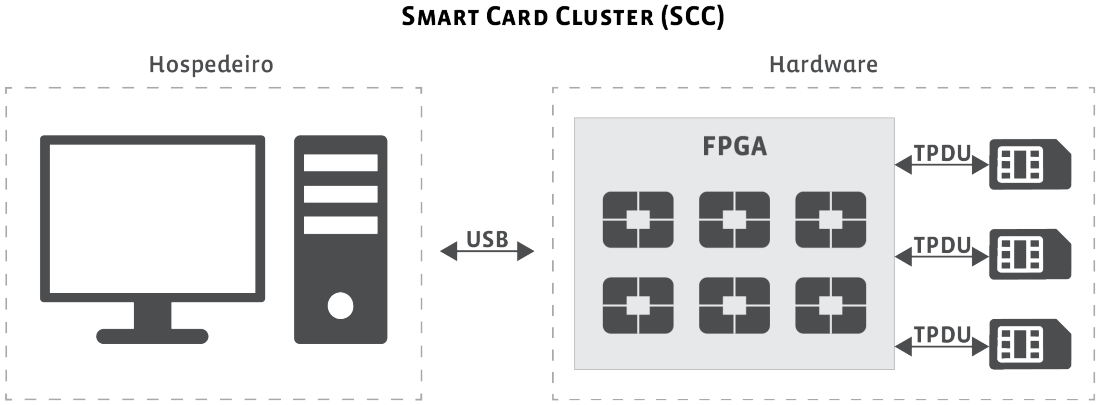


Figura 4.1: Arquitetura geral do SCC.

Nas Tabelas 3.1 e 3.2, as células em destaque na cor cinza claro representam onde o SCC pode ser aplicado e em cinza escuro representa o caso de uso ao qual o SCC será aplicado neste trabalho. Ele pode ser aplicado tanto na camada de aplicação como na camada de rede e é capaz de ser integrado no dispositivo, no *gateway* ou no servidor IoT. Para o caso de uso que será apresentado, ele é utilizado na camada de rede e aplicado ao *gateway*.

Com exceção da estrutura de dispositivos e do servidor IoT, o SCC pode ser utilizado para assegurar qualquer um dos elementos listados na Tabela 3.2. Isso se dá, principalmente, pela utilização de SCs, que oferecem procedimentos de segurança, como autenticação, encriptação e assinatura, associados a computação resistente a ataques, que é feita em hardware. O SCC pode agregar segurança por meio de autenticação, armazenamento seguro, assinatura, dispositivo embarcado, encriptação e geração de chaves. Especialmente, o SCC está entre os poucos trabalhos que utilizam dispositivos embarcados e geram chaves seguras, e também pode ser utilizado para assegurar a autenticidade de serviços e de usuários. Neste trabalho o SCC é utilizado para gerar assinaturas, acrescentando, com isso, autenticidade e integridade às mensagens por meio da integração deste dispositivo ao *gateway*.

A proteção para alguns dos ataques apresentados por [53], descritos brevemente na Seção 2.2, pode ser feita pelo SCC. Utilizando os elementos seguros com estratégias de autenticação, encriptação ou assinatura, o SCC pode mitigar ataques baseados na comunicação ou identidade, dentre eles estão o *Sybil*, o *Man-in-the-middle* e o *Replay*. Os ataques que o SCC não consegue resolver são aqueles baseados em estratégias de ataques físicos, os baseados em localização interna a rede, e os baseados em hospedeiros previamente autorizados.

A utilização de SCs faz com que o sistema possa gerar e armazenar chaves criptográficas, de forma que elas nunca sejam expostas. Da mesma forma, dados podem ser armazenados neles, para terem um maior nível de segurança. Eles também podem ser utilizados para controle de acesso e gerenciamento de autenticidade, tão como para encriptar e assinar mensagens.

O SCC une as vantagens da utilização de SCs com as dos FPGAs, para oferecer uma plataforma portátil, escalável, modular, compatível com protocolos mais difundidos e que tenha o custo apropriado para aplicações de pequeno e grande porte. Essas características do sistema proposto são descritas a seguir:

- **Portabilidade:** Permite que o sistema possa ser implantado em diversos FPGAs. Assim, o usuário pode adquirir o FPGA que seja suficiente para a quantidade de SCs desejados. Isso evita desperdício de recursos. Além disso, os próprios SCs tem a portabilidade como característica, permitindo que eles sejam facilmente incluídos, configurados e trocados, se houver alguma necessidade.

- **Escalabilidade:** É possível expandir o número de SCs utilizados, para atender a necessidade do usuário. Se o limite de SCs em um mesmo FPGA for alcançado, a quantidade de FPGAs também pode ser expandida, ou o FPGA pode ser trocado por outro que tenha maior capacidade. Com o SIM Array, apresentado na Seção 2.3.4, a expansão se dá de 32 em 32 *slots* de SCs, pois cada SIM *Board* dele já vem estruturado com essa quantidade mínima. Para algumas aplicações isso pode ser desnecessário, ou inviabilizar sua utilização, devido ao alto custo.

- **Modularidade:** Cada módulo do sistema é independente e facilmente alterável. Deste modo, é possível mudar o conector USB, incluir mais de um FPGA, variar o

computador hospedeiro, evoluir o sistema para protocolos de transmissão posteriores, incluir novos comandos ao conjunto do CCID implementado e colocar circuitos de interface para ler SCs no formato de cartões de crédito, ou SIM *Cards*.

- **Compatibilidade:** O sistema é desenvolvido se baseando em padrões de comunicação abertos e dispositivos embarcados bem estabelecidos e difundidos, como o PC/SC, o CCID e o ISO/IEC 7816-3 [32]. Isso facilita a evolução do sistema, a integração em diversos ambientes e possibilita a replicação do sistema com componentes acessíveis.

- **Custo:** O custo do sistema depende da necessidade de SCs do projetista. Como a plataforma é modular, há liberdade de incluir módulos com maior ou menor capacidade de processamento para suportar o número de SCs desejados, e que estejam dentro do orçamento esperado.

Com esses atributos o SCC consegue ser distribuído entre vários nós na rede, como dispositivos, *gateways* e servidores IoT. A forma que ele deve ser distribuído depende do nível de proteção que as informações e dispositivos precisam, e dos recursos financeiros e estruturais que podem ser utilizados na rede a ser assegurada.

A Figura 4.2 mostra o relacionamento entre os módulos do sistema. Por ela é possível perceber que o SCC é dividido em duas partes: o hospedeiro e o hardware. O hospedeiro é onde são implementados os softwares necessários ao sistema, nele está inserido o *framework* PC/SC e o *driver* de dispositivo, que permite comunicação USB com o FPGA. Ele propicia também, a implementação de aplicações de alto nível, para integrar os recursos do hospedeiro com outros programas e gerenciar a interface de rede. Analisando o hardware, um FPGA é utilizado para fazer o gerenciamento das leituras e distribuição das mensagens entre os SCs. A interface é um circuito eletrônico desenvolvido para fazer a conversão dos níveis de tensão entre o FPGA e o SC. O Conector hospedeiro faz o gerenciamento da comunicação física entre o FPGA e o *driver* no computador hospedeiro.

4.2 Cenário de aplicação do SCC

Em IoT, dispositivos heterogêneos enviam dados para um servidor, que os armazena no banco de dados, e os tornam disponíveis para serem consultados por usuários e processados por aplicações. Lá ocorre algum processamento, em seguida comandos podem ser enviados para os dispositivos conectados na rede. O gerenciamento de todo esse processo é feito pelo *middleware* IoT, o qual garante o funcionamento adequado e abstrai os protocolos de comunicação utilizados e o tratamento interno dos dados.

Os *gateways* são necessários devido a heterogeneidade das informações que os dispositivos inserem na rede. As empresas distribuidoras podem definir formatos de mensagens diferentes ou utilizar protocolos de comunicação variados, como ZigBee, TCP/IP, RFID ou IEEE 802.11.

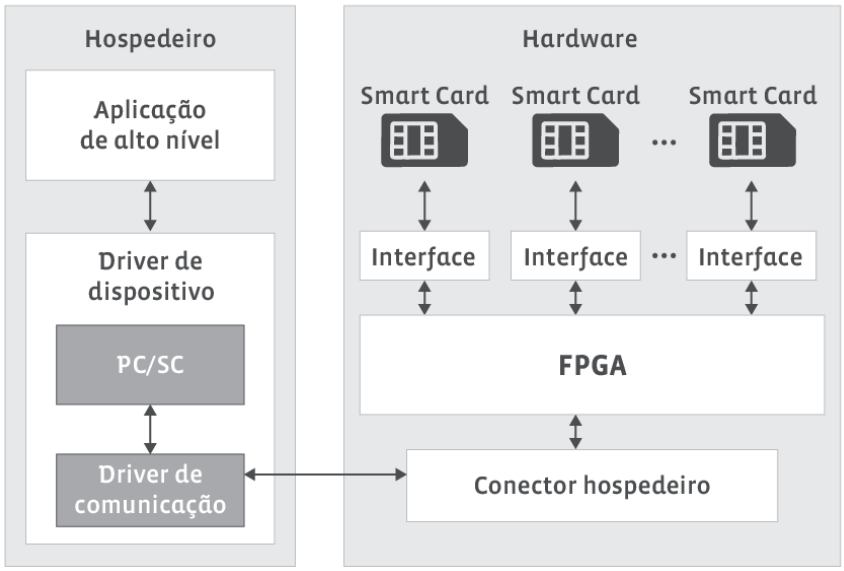


Figura 4.2: Arquitetura interna do SCC, mostrando o relacionamento entre os módulos.

Assim, o *gateway* faz a transição entre equipamentos com o mesmo protocolo de comunicação e o protocolo de troca de mensagens utilizado pelo *middleware*.

O cenário onde o SCC será aplicado, pode ser visualizado pela Figura 4.3. Ele representa uma universidade pública em que vários dispositivos de medição localizados em um prédio (Prédio medido) enviam os valores medidos para outro prédio (Prédio de controle). Os dispositivos no Prédio medido são agrupados pelo protocolo de transmissão, formato dos dados ou tipo dos dispositivos. Assim, para cada grupo de dispositivos há um *gateway*, que agrega as mensagens deles e transmite para o *middleware* IoT, presente no Prédio de controle. Quando o *middleware* IoT deseja se comunicar com algum dispositivo, os dados são enviados para o *gateway* responsável, que os redirecionam para o dispositivo, utilizando o protocolo e formato de mensagem adequados. Há uma distância grande entre prédios, da ordem de centenas de metros ou quilômetros, enquanto há uma pequena distância entre dispositivos e o *gateway* atrelado a eles, no máximo algumas dezenas de metros, pois devem estar no mesmo prédio.

A grande distância entre prédios traz mais possibilidades de ataques *Man-in-the-middle*, já que os dados trafegam em espaço comum, que qualquer pessoa tem acesso. Entre os dispositivos e o *gateway* há um nível maior de segurança, devido aos prédios ficarem fechados durante a noite e possuir proteção local institucional. Neste cenário apenas pessoas autorizadas acessam os prédios.

Para este cenário, não é necessário que os dados das medições sejam confidenciais, já que são informações de um órgão público, podendo estar disponíveis para uso coletivo. Porém, a autenticidade e integridade das mensagens é crucial, uma vez que não devem haver informações adulteradas, replicadas, ou de autoria duvidosa.

A proposta é utilizar o SCC para assegurar a comunicação entre o *middleware* e os *gateways*, garantindo que as mensagens trocadas entre eles sejam assinadas pelos elementos

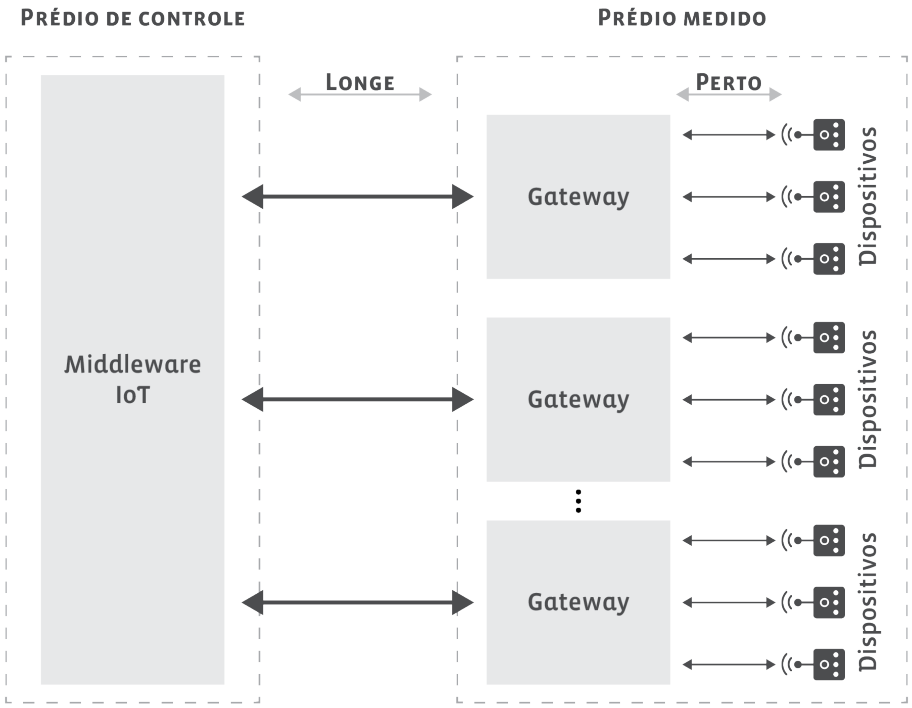


Figura 4.3: Visualização do cenário de IoT.

seguros, com as chaves geradas nos próprios SCs. Isso impede a inserção de mensagens indevidas e garante a autoria e não repudio das mensagens. Na arquitetura projetada, os *gateways* fazem a separação entre os grupos de segurança.

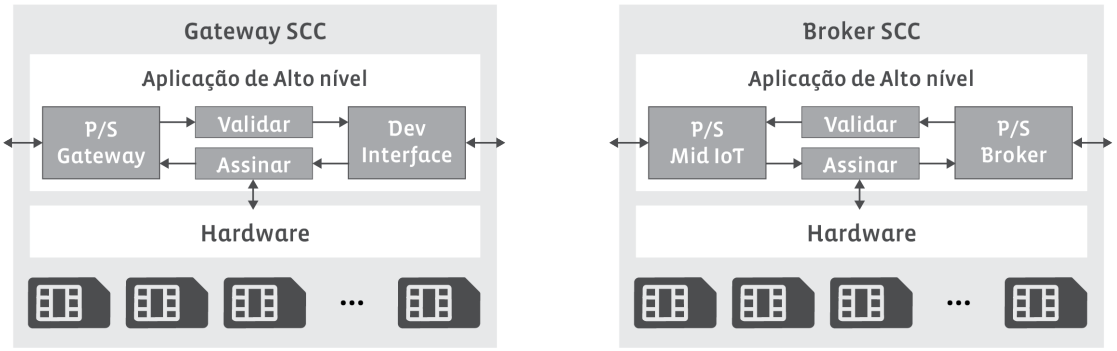
O protocolo de troca de mensagens utilizado entre o *middleware* IoT e os *gateways* é o *Message Queue Telemetry Transport* (MQTT), que é do tipo Publicador/Subscritor (*do inglês Publish/Subscribe - P/S*). Ele é interessante pois o publicador é desacoplado do subscritor, isto é, há um elemento entre eles que concentra a recepção e faz a distribuição das mensagens. O componente responsável por essa tarefa é chamado de *broker*.

Tanto o *gateway* como o *broker* são componentes que tornam a troca de mensagens transparente, para as partes comunicantes, no caso, o *middleware* IoT e os dispositivos. A arquitetura proposta busca atuar nesses componentes para manter a compatibilidade entre as partes, sem alterações bruscas. Com isso são introduzidos os elementos: Gateway SCC e Broker SCC. Eles são plataformas SCC que possuem como aplicação de alto nível as configurações do *gateway* e do *broker* do sistema IoT. Cada Gateway SCC e Broker SCC deve ser analisado para mensurar a quantidade de SCs que eles devem possuir. Essa quantidade depende da demanda de dispositivos para os Gateways SCC, e da frequência de requisições feitas pelo *middleware* IoT para o Broker SCC. A seguir, é descrito o comportamento interno deles:

- **Gateway SCC:** A Figura 4.4(a) indica como ele se comporta. Os dispositivos conectados enviam as medições para o Gateway SCC, que por meio da interface com o dispositivo (Dev Interface) processa o protocolo de comunicação e o formato da

mensagem utilizado por ele. Então, a mensagem é assinada pelo hardware seguro e repassada para P/S Gateway transmitir para o Broker SCC. Quando uma mensagem é recebida do Broker SCC, a assinatura é verificada, se for válida, os dados são colocados no formato que os dispositivos entendem e transmitidos, se não for válida ela é descartada.

- **Broker SCC:** O Broker SCC está representado na Figura 4.4(b). Ele recebe as publicações do *middleware* IoT, assina através do hardware seguro e transmite para os Gateways SCC, que tenham assinado o tópico em questão. Ao receber a publicação vinda de algum Gateway SCC, o Broker SCC verifica se a assinatura é valida, e repassa ela ao *middleware* IoT se estiver segura, ou descarta, se não estiver.



(a) Gateway SCC (b) Broker SCC

Figura 4.4: Diferentes configurações do SCC para uma arquitetura IoT.

A Figura 4.5 mostra o cenário apresentado ao incluir os SCCs. Agora o sistema assegura a comunicação entre *middleware* e *gateways* impedindo que invasores consigam inserir, alterar, repetir mensagens ou se passar por algum dispositivo na rede.

Um exemplo do processo completo de troca de mensagens entre o *middleware* IoT e o dispositivo é apresentado na Figura 4.6. Inicialmente, o *middleware* IoT se inscreve no tópico G1, que é o tópico referente às informações enviadas pelo Gateway SCC 1. O Gateway SCC 1 se inscreve no tópico S1, esse é o tópico no qual o *middleware* IoT publica os comandos a serem enviados para o Gateway SCC 1. O dispositivo envia as medições M1 e M2 para o Gateway SCC 1, onde elas são assinadas, e publicadas no tópico G1 para serem enviadas para o Broker SCC como uma única mensagem, já com a assinatura (SIG). O Broker SCC verifica a assinatura, se estiver correta, repassa a mensagem para o Middleware IoT, que está subscrito no tópico G1. Em seguida, o *middleware* IoT envia o comando C1 publicando no tópico S1. O Broker assina C1 e transmite para o Gateway SCC 1, por ele estar inscrito no tópico S1. O Gateway SCC 1 verifica a assinatura de C1 e então envia o comando verificado para o dispositivo.

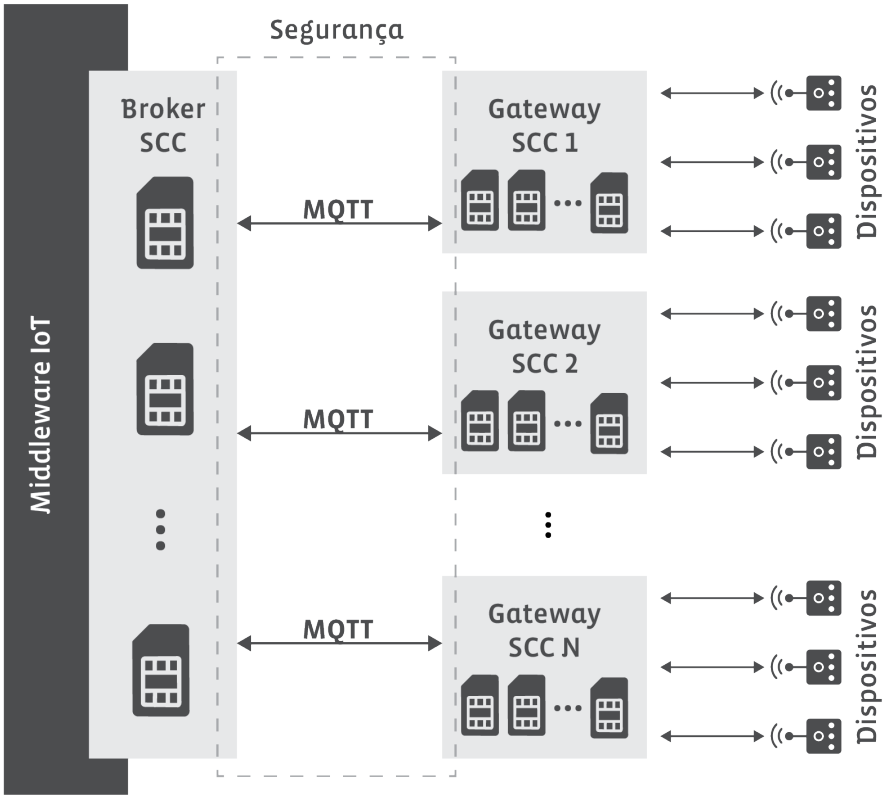


Figura 4.5: Cenário de IoT assegurado com o SCC.

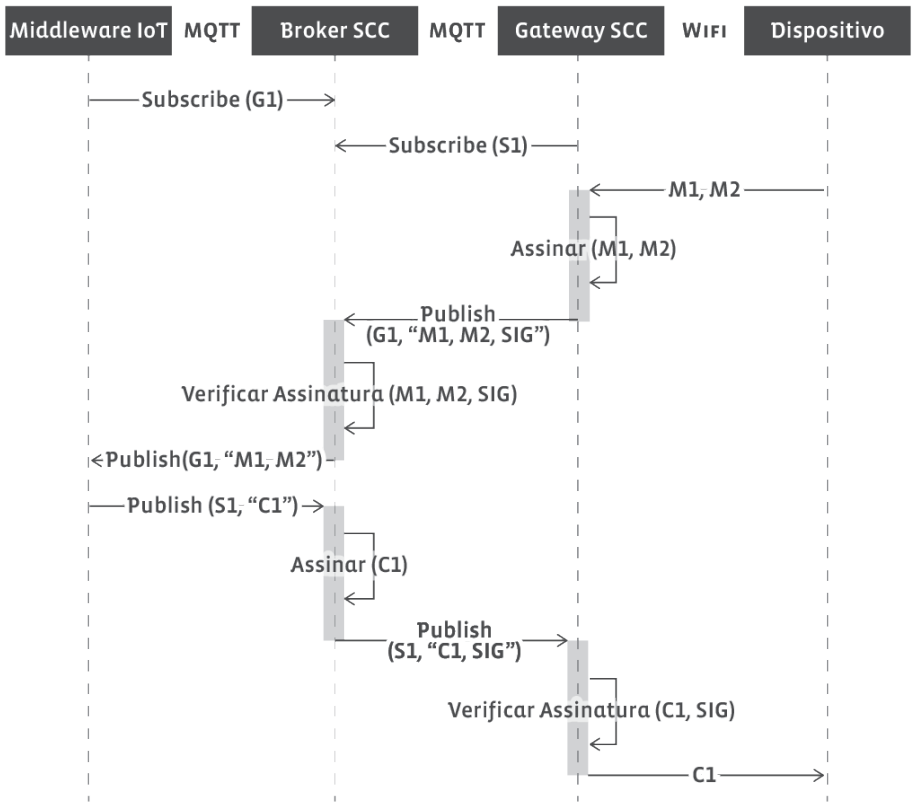


Figura 4.6: Exemplo de sequência de troca de mensagens para o cenário de IoT, com o Gateway SCC e o Broker SCC.

4.3 FPGA

O FPGA faz o gerenciamento da comunicação entre vários SCs e um computador hospedeiro. Para alcançar esse objetivo, foi feito uso da arquitetura de implementação em FPGA apresentada na Figura 4.7. A arquitetura consiste de dois módulos principais: o FIIP, para interface direta com cada SC, e o Controlador de Comunicação, responsável pela troca de informações com o Conector hospedeiro. Associado a cada módulo FIIP há dois *buffers* (RD Buffer SC e WR Buffer SC) com capacidade de 512 bytes cada e estrutura de dados Primeiro a Entrar, Primeiro a Sair (*do inglês First In, First Out - FIFO*). Eles servem para fazer a transição das mensagens entre os dois módulos principais. A seguir, os módulos FIIP e Controlador de Comunicação, serão detalhados.

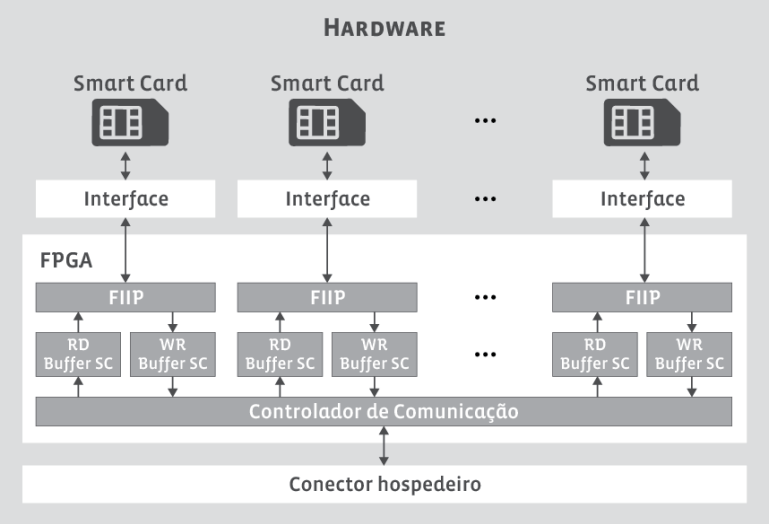


Figura 4.7: Arquitetura da implementação em FPGA.

4.3.1 FIIP

O módulo FIIP é responsável por controlar os SCs, se comunicando diretamente com o módulo externo Interface e com os *buffers* internos de leitura e escrita, o RD Buffer SC e o WR Buffer SC. O FIIP representa a implementação dos processos retratados no protocolo ISO/IEC 7816-3 [32]. Com isso ele visa controlar o processo de inicialização, o recebimento da ATR e a transferência dos bytes utilizando as TPDUs T=0 e T=1. A arquitetura de implementação em FPGA do módulo FIIP está disponível na Figura 4.8, porém ela foi simplificada para melhor compreensão da estrutura. Os seus 11 submódulos serão detalhados a seguir:

1. **Clock:** Converte o clock de operação do FPGA para o clock aceito pelo SC. O módulo considera que o FPGA fornece uma entrada de 50MHz em *FPGA Clock*. Então, ele considera o fator de divisão que vem no fio *Clock value* e faz a redução da frequência de entrada, para o valor que é aceito pelo SC. No recebimento do ATR é aplicado 5MHz como saída para o SC em *SC Clock*, mas pode ser alterado de

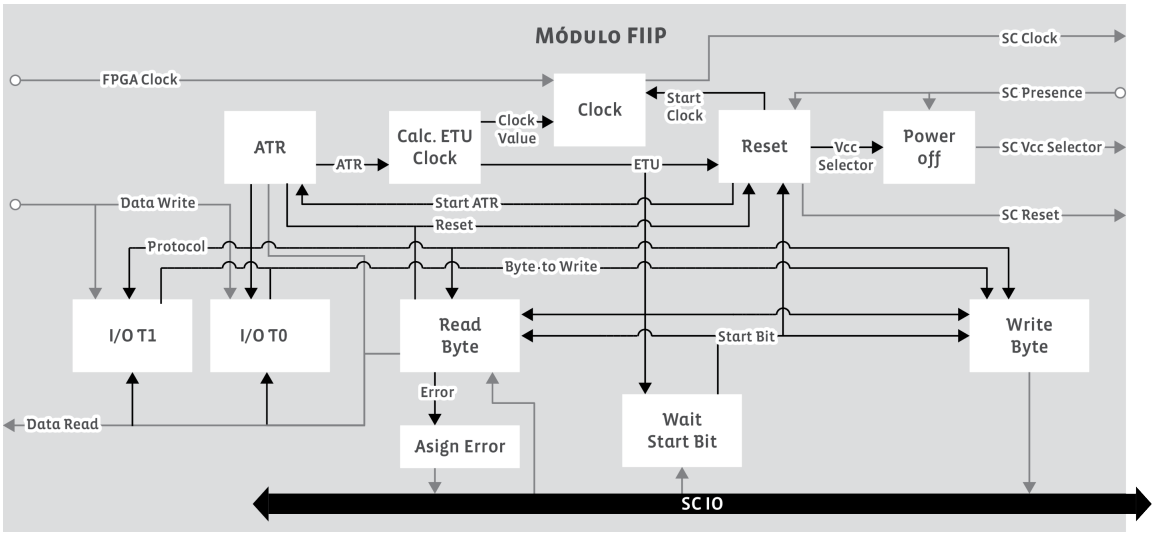


Figura 4.8: Arquitetura e conexões internas do módulo FIIP.

acordo com a preferência do SC. Os outros módulos internos ao FIIP são operados na frequência liberada por este módulo.

2. **Power Off:** Executa a rotina de desligamento do SC, caso haja uma desconexão manual recebida em *SC Presence*. Isso evita que desligamentos repentinos causem danos ao SC. A rotina de desligamento segue a seguinte sequência: primeiro a transferência de dados é imediatamente encerrada, em seguida o clock e o reset são desligados, só então o Vcc será suspendido. Esse módulo também intermedeia o envio do sinal que seleciona o Vcc (1.8V, 3V ou 5V) a ser liberado para o SC no fio *SC Vcc Selector*. Contudo, quem de fato seleciona o Vcc é o módulo Reset que informa o módulo Power Off por meio do fio *Vcc Selector*.

3. **Reset:** Realiza o processo de inicialização do SC, aguardando os ciclos necessário para liberar cada sinal no momento adequado, e seleciona o Vcc (1.8V, 3V ou 5V) a ser liberado para o SC. Primeiro, o Vcc de 1.8V é fornecido ao SC, em seguida o clock é liberado em *Start Clock* e então é colocado o sinal de reset no fio *SC Reset*.

   O módulo Reset aguarda o recebimento do bit que sinaliza o início de transmissão do ATR (*Start Bit*). Se não for recebido no tempo limite, o processo de reset volta ao inicio, mas agora o Vcc de 3V é escolhido. Se mais uma vez o ATR não chegar, há outro reinicio do processo, dessa vez com o Vcc de 5V. Esse processo ocorre para que seja aplicado o Vcc ideal ao SC utilizado. Considera-se que o SC esteja com defeito quando nenhuma das 3 tensões resultar em resposta.

   Havendo o recebimento do sinal de inicio do ATR em *Start Bit*, o módulo de ATR é alertado pelo fio *Start ATR* e o processo de leitura prossegue.

4. **Read Byte:** Faz o processamento dos bytes recebidos do SC pelo canal *SC IO*. Ele obtém cada bit, agrupa em um byte e envia pelo fio *Data Read*. Outra tarefa é verificar

o bit de paridade, com o intuito de saber se o byte chegou corretamente.

Também é responsável por sinalizar se o byte não foi recebido no tempo máximo de leitura. Quando isto ocorre, é dito que o *card* está não responsivo, então o módulo Reset é alertado em pelo fio *Reset*, para que haja a reinicialização do SC.

Para cada byte o SC envia 10 bits. O primeiro bit, que é chamado de start bit, sinaliza o início da transmissão, dado que a transmissão é assíncrona. Os 8 bits seguintes, são o byte esperado. O último é o bit de paridade, utilizado para verificação de erro na sequência de bits recebidos. Em seguida, há um tempo de espera, que pode ser utilizado para sinalizar o SC se ocorrer erro na recepção dos byte. Quando ocorre erro, o sinal é colocado no fio *Error*.

5. **Write Byte:** Atua de forma inversa ao que ocorre com o módulo Read Byte. O módulo Write Byte recebe os bytes pelo fio *Byte to write* (vindo dos módulos I/O T0 e I/O T1) e transmite para o canal *SC IO*. Para isso ocorrer, ele serializa o envio de cada bit, incluindo o start bit no início e calculando o bit de paridade.

   Após o envio, um tempo é aguardado para recebimento de um possível sinal de falha na transmissão. Ocorrendo a falha, o byte é reenviado. Se não for constatado erro na transmissão, o processo de escrita do byte é finalizado e o módulo que requisitou o envio é alertado.

6. **ATR:** Seu objetivo é realizar o mecanismo de leitura e validação dos bytes do ATR. Ele é inicializado pelo módulo Reset, através do fio *Start ATR*, e recebe os bytes pelo fio *Data Read*.

   Primeiro ele processa o byte TS, e configura a leitura do módulo Read Byte para a convenção de bits definida no recebimento desse byte. Em seguida recebe o byte T0, que indica a quantidade de bytes de histórico disponíveis e se os bytes TA1, TB1, TC1 e TD1 estão inclusos na ATR. Os bytes TAi, TBi, TCi e TDi vão sendo processados de acordo com a sinalização de suas presenças no byte TD(i-1). Em seguida, os bytes de histórico são lidos. Por último, se estiver incluso, o caractere de checagem (TCK) é lido, e através dele é verificado se a ATR foi recebida corretamente.

   Se algum erro foi detectado, é enviado um sinal no fio *Reset* e a leitura é reiniciada, se ocorrer mais uma vez esse erro, o SC é considerado não responsivo. Ao receber a ATR sem falhas, ele é enviado por meio do fio *ATR* para o módulo Calc ETU Clock.

   Os parâmetros de configuração recebidos nestes bytes ajustam o funcionamento interno do módulo FIIP, de forma que os módulos internos agora funcionem com a frequência de clock escolhida, e sejam liberados para o SC a tensão e o *clock* preferidos dele.

7. **Calc ETU Clock:** Calcula a Unidade de Tempo Elementar (*do inglês Elementary Time Unit - ETU*), que representa a quantidade de ciclos necessários para leitura/escrita completa de um bit. O resultado é compartilhado com os módulo que fazem tratamento de bits.

   Este módulo também informa ao módulo Clock através do fio *Clock Value* a relação entre o *clock* de entrada do FPGA (*FPGA Clock*) e o clock desejado pelo SC (*SC Clock*). Assim, de acordo com o valor recebido no ATR pelo módulo ATR, esse cálculo é ajustado fazendo uma relação de quantos pulsos de clock recebidos na entrada são equivalentes a um pulso que deve ser colocado na saída.

8. **I/O T0:** Este módulo faz o processamento da APDU e a envia para o SC de acordo com o protocolo de transmissão T=0. Ele só é ativado se receber do módulo ATR, através do fio *Protocol*, que o protocolo a ser utilizado é o T=0. Ele repassa os bytes a serem escritos para o módulo Write Byte pelo fio *Byte to Write* e recebe os bytes lidos do módulo Read Byte no fio *Data Read*.

   Primeiro são processados e enviados os bytes do cabeçalho. Em seguida, se houver, são enviados os bytes de dados. A resposta do SC é, então, recebida e encaminhada para o buffer de recepção.

9. **I/O T1:** De forma similar ao módulo I/O T0, esse módulo faz a interface de troca de mensagens, porém seguindo o protocolo de transmissão T=1. Os módulos Write Byte e Read Byte são utilizados para enviar e receber os bytes, respectivamente. Como no módulo I/O T0, esse módulo só funciona se for recebido do módulo ATR, através do fio *Protocol*, que o protocolo a ser utilizado é o T=1.

   A APDU recebida no buffer de entrada é enviada, em seguida são recebidos os bytes do *card* respeitando a separação entre blocos, o BWT e o BGT. Se algum desses tempos for extrapolado, um erro é sinalizado e a escrita é reiniciada.

10. **Assign Error:** Quando um erro na verificação do bit de paridade é detectado pelo módulo Read Byte, o módulo Assign Error é alertado pelo fio *Error*. Este módulo sinaliza o SC com um bit pelo canal de I/O, no período de tempo exato em que o SC está esperando essa sinalização.

11. **Wait Start Bit:** É utilizado para sinalizar o início do recebimento de um novo byte. Os módulos Reset, Read Byte, Write Byte e I/O T1 têm interesse nessa informação.

    Ele monitora constantemente o canal *SC IO*. Quando percebe a chegada do *start bit*, é feito o processo de *triple sample*. Esse processo representa a coleta de três amostras de tensão em *SC IO* para o mesmo bit, em diferentes momentos no intervalo de um ETU. Isso garante que variações repentinas na tensão não causem leitura errada do

bit. Verificando que a recepção foi correta, os módulos interessados são sinalizados por meio do fio *Start Bit*.

4.3.2 Controlador de Comunicação

O módulo Controlador de Comunicação está presente na Figura 4.9. Ele faz o gerenciamento dos dados que são enviados ou recebidos pela interface USB. Dessa forma, ele distribui entre os SCs as informações específicas recebidas do computador hospedeiro, colocando os dados nos *buffers* de recepção (RD Buffer SC) associados a cada módulo FIIP. Os dados que os SCs respondem são conduzidos ao buffer de escrita (WR Buffer SC) pelo módulo FIIP. Dessa maneira, o módulo Controlador de Comunicação assegura que os dados em WR Buffer SC sejam enviados para o computador hospedeiro pela interface USB.

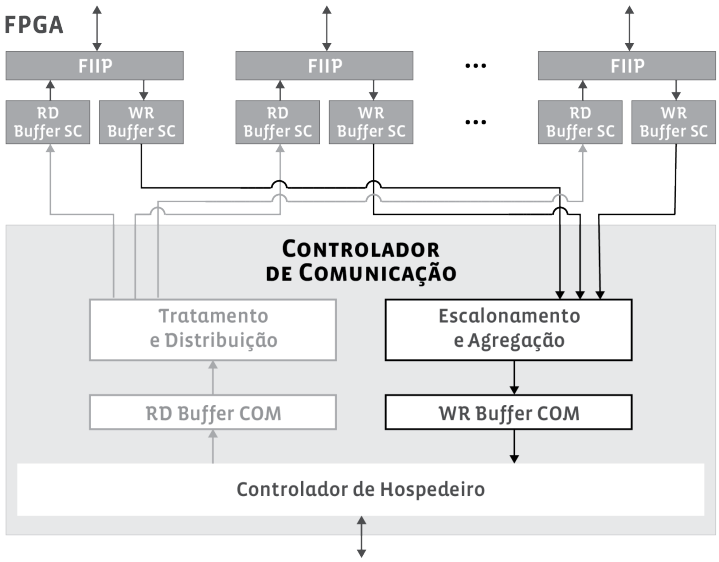


Figura 4.9: Arquitetura do módulo de comunicação.

A comunicação com o hospedeiro é baseada no protocolo CCID, descrito na seção 2.4.2. Apesar de não haver compatibilidade total com este protocolo, alguns dos seus comandos e o formato das mensagens foram implementados no Controlador de Comunicação. A Tabela 4.1 descreve os comandos que este módulo é capaz de processar.

Tabela 4.1: Comandos CCID implementados.

Comando	**Descrição**
IccPowerOn	Sinaliza para o SC ser iniciado e espera como retorno o ATR
PowerOff	Informa ao leitor que o *card* deve ser desconectado
GetSlotStatus	Faz a requisição do estado atual de presença do SC
XfrBlock	Transmite blocos de dados
GetParameters	Solicita a informação do protocolo de transmissão aceito

O Controlador de Comunicação possui três submódulos: um módulo que controla a comunicação com o computador hospedeiro (Controlador de Hospedeiro), um módulo de tratamento de comandos e distribuição de mensagens entre os SCs (Tratamento e Distribuição) e um módulo que escalona e agrega as respostas de cada SC (Escalonamento e agregação). Além de tais submódulos, há dois *buffers*, um para armazenar as mensagens que chegam (RD Buffer COM) e outro para colocar as mensagens que serão enviadas (WR Buffer COM). Cada um deles possui capacidade para armazenar até 4096 bytes. Os submódulos serão melhor explicados a seguir:

1. **Controlador de Hospedeiro:** Controla a transmissão de dados entre o FPGA e o hospedeiro. Quando o hospedeiro envia informações, este módulo lê e repassa para RD Buffer COM. Se houver dados em WR Buffer, eles são enviados para o Conector hospedeiro. Esse módulo é diretamente influenciado pelo Conector hospedeiro a ser utilizado. O conector utilizado nessa implementação será relatado na Seção 4.5. Conectores diferentes requerem implementações diferentes desse módulo.

2. **Tratamento e Distribuição:** Realiza a leitura dos comandos recebidos em RD Buffer COM, e avalia se estão entre os comandos na Tabela 4.1 e se o formato da mensagem está de acordo com o da Tabela 2.5. Após reconhecer o comando que foi enviado pelo hospedeiro, o procedimento que ele requer é desempenhado. Após o comando ser completamente processado, o módulo Escalonamento e Agregação é alertado, para que possa transmitir a resposta quando ela estiver disponível.

3. **Escalonamento e Agregação:** Faz o escalonamento da leitura entre os diversos *buffers* WD Buffer SC, que possui as respostas dos SCs, para enviar para o *buffer* WD Buffer COM. Para isso, este módulo avalia a cada ciclo, se o módulo FIIP associado a cada um dos SCs escreveu completamente a resposta do SC no *buffer* WD Buffer SC. Quando a resposta está neste *buffer*, o cabeçalho do comando CCID é construído e escrito em WD Buffer COM junto com os dados que vêm de WD Buffer SC. Terminando a leitura, outro módulo FIIP é avaliado, e assim continua. Todas as mensagens de resposta que foram inseridas em WD Buffer COM ficam disponíveis para que o Controlador de Hospedeiro faça a transmissão, quando o canal de comunicação (USB) com o hospedeiro esteja disponível.

4.4 Interface

Para haver a troca de informações entre o FPGA e o SC, é necessário um circuito que faça a interface elétrica entre eles. Com essa finalidade o módulo Interface, na Figura 4.2, representa o circuito eletrônico utilizado para fazer essa conversão. Ele é essencial para dispor os corretos valores de tensão em cada contato do SC, como relata [59].

O circuito eletrônico de leitura de SCs, e a disposição dos contatos que fazem a interface com o FPGA, podem ser vistos na Figura 4.10. A Tabela 4.2 traz uma breve descrição de cada um dos pinos.

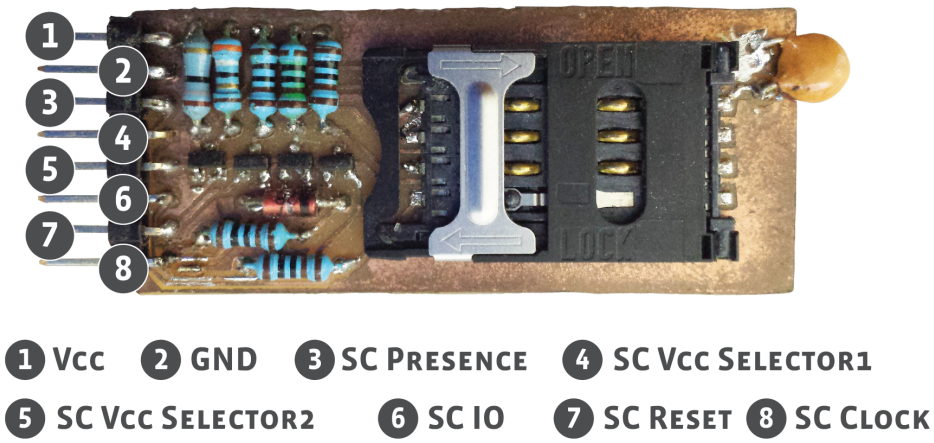


Figura 4.10: Circuito eletrônico de interface entre FPGA e SC.

Tabela 4.2: Descrição dos pinos do circuito de leitura.

	Pino	**Descrição**
1	Vcc	Tensão de alimentação da placa e do *card*.
2	GND	Sinal de conexão com o terra.
3	SC *Presence*	Indica para o FPGA se o *card* está conectado.
4	SC Vcc *selector* 1	Reservado para uso futuro.
5	SC Vcc *selector* 2	Ativa ou desativa o Vcc para o SC.
6	SC IO	Canal de troca de dados entre o *card* e o FPGA.
7	SC *Reset*	Usado para enviar o sinal de *reset* para o *card*.
8	SC *Clock*	Pino para envio do *clock* de operação do *card*.

Este circuito faz a conversão entre os níveis de tensão de 3.3V, utilizado no I/O do FPGA, e 5 ou 3V dos SCs, que são, respectivamente, as classes A (5V) e B (3V) de SCs, segundo a Tabela 2.1. Este dispositivo não inclui a classe C (1.8v), por ser mais recente e ter seu escopo limitado às comunicações móveis [59]. Esta classe será atendida em trabalhos futuros.

A partir do FPGA, o módulo FIIP envia os sinais de *reset*, *clock* e ativação do Vcc, nos respectivos pinos: *SC Reset*, *SC Clock* e *SC Vcc selector 2*. Para fazer a conversão do nível de tensão dessas saídas do FPGA, são utilizados transistores de efeito de campo (MOSFETs), que foram montados em configuração fonte comum, com um resistor de *pull-up*. Os transistores operam como chaves, de forma que ative ou desative a tensão correta em cada um dos contatos

do SC, como demonstra a Figura 4.11. Os sinais são inseridos em *FPGA out* e as respostas estão em *Card in*.

O I/O precisa de uma atenção especial, uma vez que ocorre conversão da tensão em ambos os sentidos. Portanto, também deve haver conversão da tensão de operação do SC, para a tensão do FPGA. A Figura 4.12 mostra como o mesmo transistor MOSFET deve ser conectado para obtermos os resultados esperados. Essa configuração é a mesma utilizada para barramentos do tipo Circuito Inter-Integrado (*do inglês Inter-Integrated Circuit - I2C*), relatados em [64].

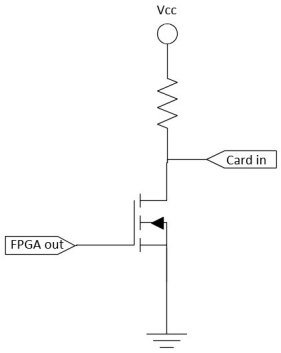


Figura 4.11: Configuração para conversão dos sinais de *reset*, *clock* e ativação do vcc, que saem do FPGA (FPGA out) e entram no SC (Card in).

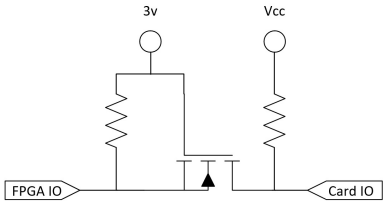


Figura 4.12: Configuração para conversão do sinal de I/O entre o FPGA e o SC.

4.5 Conector hospedeiro

A transmissão física de dados entre o hospedeiro e o FPGA é feita por meio do Conector hospedeiro. Ele é integrado ao FPGA para converter o protocolo utilizado pelo computador hospedeiro em sinais que possam ser captados pelas interfaces de I/O do FPGA. Esse conector pode ser trocado para atender diferentes protocolos de comunicação, como USB, RS-232, I2C, entre outros.

Para este trabalho a tecnologia USB foi escolhida, por oferecer imediato reconhecimento após o dispositivo ser conectado, a alta taxa de transmissão e a fácil compatibilidade com os computadores atuais. O dispositivo FTDI FT245 [24] é utilizado para atender esse propósito. Ele implementa o protocolo USB 2.0 para tratar os dados recebidos e enviados. Este conector possui taxa de transmissão de 1 MB/s, o seu *buffer* de recepção suporta 128 bytes e o *buffer* de escrita provê armazenamento para 256 bytes. A comunicação com o FPGA é feita por interface paralela de 8 pinos.

A transferência dos dados entre o conector e o FPGA é feita por meio das Entradas e Saídas de Propósito Geral (*do inglês General Purpose Input/Output - GPIO*). Quando recebe dados do hospedeiro, o Conector hospedeiro os coloca em um *buffer* de recepção interno, o FPGA é alertado e ele espera até o FPGA fazer a leitura dos dados. Para o FPGA enviar algum dado, deve analisar se o Conector hospedeiro está em um estado que permite a recepção, só então os dados podem ser enviados para o *buffer* de recepção do Conector hospedeiro.

4.6 **Driver de dispositivo**

O Driver de dispositivo agrega os módulos PC/SC e Driver de comunicação. O PC/SC (descrito na Seção 2.4.3) define uma API de alto nível para os programas do usuário, e outra interface de baixo nível que gerencia os *drivers*. O Driver de comunicação faz a transição dos dados vindos do PC/SC para interface de comunicação física utilizada. Neste trabalho ele controla o conector FT245 que utiliza comunicação USB, como foi delineado na Seção 4.5.

A Figura 4.13 mostra o comportamento do Driver de comunicação. Os comandos recebidos do módulo PC/SC são enviados, um por vez, para cada SC. Assim, mesmo suportando vários SCs, quando um comando é enviado para o FPGA, o *driver* espera a resposta a esse comando para só então enviar o próximo comando, com isso, quando um SC está executando, os outros estão em estado de espera.

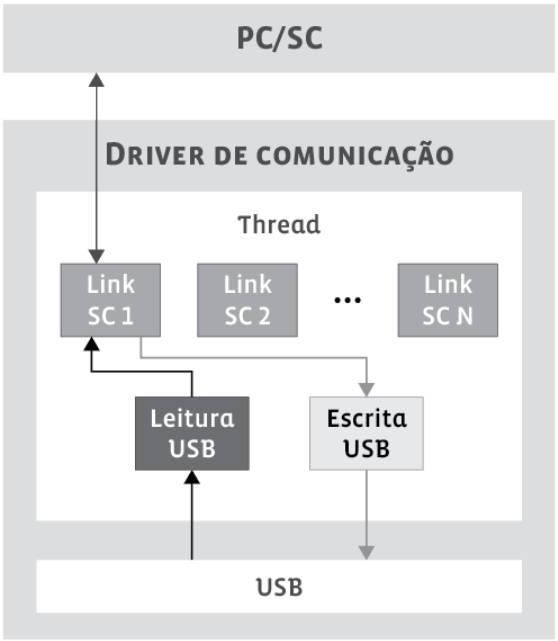


Figura 4.13: Driver de comunicação implementado para o SCC.

5

RESULTADOS E ESTUDO DE CASO

Este capítulo é dedicado a apresentar os resultados obtidos e a aplicar a plataforma proposta em um cenário específico. Para isso, na Seção 5.1 é discorrido sobre a utilização de recursos da plataforma com diferentes FPGAs. Depois, são avaliados os custos de implementação do SCC, na Seção 5.2. Em seguida, na Seção 5.3 o desempenho da plataforma é investigado. A Seção 5.4 demonstra o estudo de caso ao qual o SCC é aplicado. Por último, na Seção 5.5 são feitas as considerações finais em relação aos resultados obtidos.

A quantidade de *slots* de uma plataforma representa o número de SCs que ele pode suportar. Por isso, ao falar que o SCC possui dois *slots*, significa que ele admite até dois SCs. A quantidade de módulos "FIIP"no FPGA é equivalente a quantidade de *slots*. Porém é necessário apenas um módulo "Controlador de Comunicação"para cada FPGA.

5.1 FPGA

Nesta avaliação, o objetivo é entender se diferentes FPGAs suportam o sistema e distinguir o comportamento do SCC com eles. É analisado se os FPGAs suportam quantidades distintas de *slots* de SCs, e quais recursos do FPGA são determinantes para a quantidade máxima de *slots* que ele suporta.

Os seguintes recursos são considerados: bits de memória, Elementos Lógicos (*do inglês Logic Elements - LEs*) e blocos de Processamento Digital de Sinais (*do inglês Digital Signal Processing - DSP*). A comparação é feita para quatro diferentes Kits de Desenvolvimento (*do inglês Development Kit - DK*). A descrição de cada DK utilizado e o FPGA associado a ele, podem ser vistos na Tabela 5.1, os dados contidos nesta tabela foram tirados de [82], [3], [6], [7] e do simulador Quartus II, versão 13.1.0 [8]. O DE0-Nano e o CoreEP4CE10 são DKs de baixo desempenho e consequente baixo custo, já o Cyclone V DK e o Stratix V DK são de alto desempenho, mais caros que os primeiros e possuem blocos DSP. O custo dos DKs será demonstrado na Seção 5.2. As quatro placas utilizadas são da Altera, para não haver bruscas variações entre as tecnologias, que podem ocorrer entre placas de empresas diferentes.

A medição dos recursos de FPGA utilizados com cada um dos DKs é obtida por meio do relatório de compilação reportado pelo software de desenvolvimento Quartus II 64-bits, versão

Tabela 5.1: Placas de desenvolvimento e as características do FPGA utilizado.

	Cyclone V DK	**Stratix V DK**	**DE0-Nano**	**CoreEP4CE10**
Modelo do FPGA	Cyclone V GT	Stratix V GX	Cyclone IV E	Cyclone IV E
Referência do FPGA	5CGTFD9	5SGXEA7	EP4CE22	EP4CE10
	E5F35C7N	K2F40C2N	F17C6	F17C8
Elementos Lógicos	81920	234720	22320	10320
Frequência Máxima	125 MHz	148.5 MHz	50 MHz	50 MHz
Pinos de I/O	616	864	154	180
Bits de Memória	12492800	52428800	608256	423936
Blocos DSP	342	256	0	0

13.1.0, da Altera [8]. O código é desenvolvido com a HDL System Verilog.

São feitas várias compilações do código para cada um dos FPGAs. Em cada compilação a quantidade de *slots* é aumentada, até o Quartus II informar que o FPGA não possui recursos suficientes para a quantidade atual de *slots*. Com isso, na primeira compilação, o FPGA deve suportar um *slot*, na segunda, dois *slots*, e assim por diante. A última compilação bem sucedida, representa a quantidade máxima de *slots* que o FPGA suporta.

O Cyclone V DK suportou 32 *slots*, o Stratix V DK 25 *slots*, o DE0-Nano 5 *slots* e o CoreEP4CE10 2 *slots*. Na Tabela 5.2 é possível encontrar o consumo de LEs, bits de memória e blocos DSP, para um *slot* e para a quantidade máxima de *slots* suportada pelos FPGAs.

A Tabela 5.2 (a) compara a quantidade de LEs utilizados por cada FPGA. Para um *slot*, as placas de alto desempenho (Cyclone V DK e Stratix V DK) utilizam, praticamente, metade dos LEs das placas de baixo desempenho (DE0-Nano e CoreEP4CE10). Isso se deve a tecnologia associada, onde as primeiras utilizam quatro registradores por LE, enquanto as outras utilizam 2 registradores por LE, como é relatado em [9]. Considerando a quantidade máxima de *slots*, na mesma tabela, constata-se que as placas de baixo custo são limitas pela quantidade de LEs, enquanto isso não é um problema para as de alto custo.

A Tabela 5.2 (b) mostra que a quantidade de bits de memória consumidos varia apenas com a quantidade de *slots*, independente do FPGA utilizado. Em nenhuma das placas isto foi fator determinante para quantidade máxima de *slots* suportados.

Apenas as placas de alto desempenho possuem blocos DSP, que servem para um processamento mais preciso dos sinais digitais. A utilização de blocos DSP, registrada na Tabela 5.2 (c), é responsável pelo número máximo de *slots* suportados pela placa Stratix V DK. Embora a placa Cyclone V DK tenha utilizado 100% dos blocos DSP com 32 *slots*, não é possível avaliar além dessa quantidade, pois a descrição de hardware do SCC em FPGA é limitada a 32 *slots*.

Considerando a quantidade de SCs suportados, a placa Cyclone V GT DK apresenta o melhor desempenho. Ela é utilizada para expor os LEs dos módulos "Controle de Comunicação"e

(a) Elementos Lógicos

FPGA	**Total**	**1 *slot***		**Máximo *slots***		
		Utilizados	Uso %	*Slots*	Utilizados	Uso %
Cyclone V DK	81920	3063	4%	32	52821	64%
Stratix V DK	234720	3007	1%	25	40986	17%
DE0-Nano	22320	7014	31%	5	21811	98%
CoreEP4CE10	10320	7005	68%	2	10255	99%

(b) Bits de memória

FPGA	**Total**	**1 *slot***		**Máximo *slots***		
		Utilizados	Uso %	*Slots*	Utilizados	Uso %
Cyclone V GT	12492800	81920	<1%	32	374784	3%
Stratix V GX	52428800	81920	<1%	25	310272	<1%
DE0-Nano	608256	81920	13%	5	125952	21%
CoreEP4CE10	423936	81920	19%	2	98304	23%

(c) Blocos DSP

FPGA	**Total**	**1 *slot***		**Máximo *slots***		
		Utilizados	Uso %	*Slots*	Utilizados	Uso %
Cyclone V GT	342	17	5%	32	342	100%
Stratix V GX	256	15	6%	25	256	100%
DE0-Nano		-	-	5	-	-
CoreEP4CE10		-	-	2	-	-

Tabela 5.2: Recursos utilizados por cada FPGA, para um *slot* e para a quantidade máxima de *slots* suportada por eles.

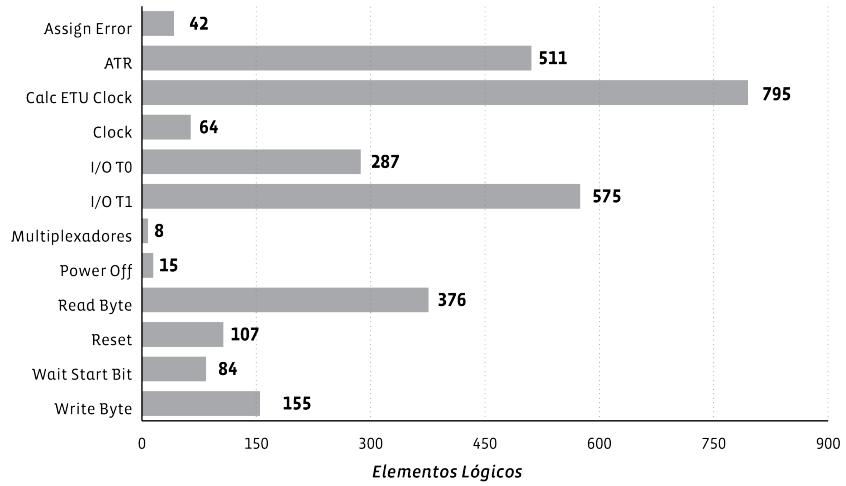
"FIIP", tão quanto cada um dos seus submódulos. O módulo "Controlador de Comunicação"consome um total de 1677 LEs, já o módulo "FIIP"consome 3033 LEs. A Figura 5.1 mostra os LEs utilizados por cada um dos submódulos.

Esse teste demostrou a portabilidade do SCC, uma vez que ele pode ser utilizado com diferentes FPGAs, para perfis de consumidores diferentes. Além disso, a capacidade de variar o número de *slots* de SCs, acrescentando mais *slots* ao mesmo FPGA, atesta a escalabilidade do sistema. Adicionalmente foi detalhado o consumo de recursos para cada módulo interno ao FPGA.

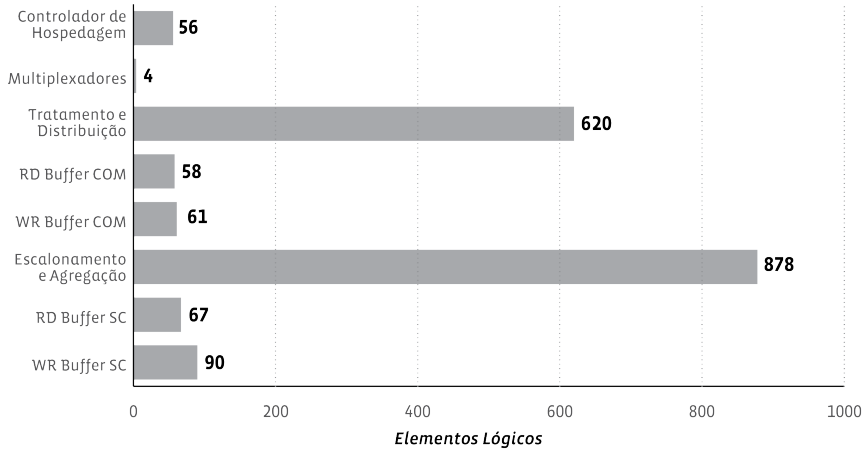
5.2 Custo

O intuito aqui é avaliar a viabilidade financeira da implementação do SCC com cada um dos quatro DKs descritos na Tabela 5.1, e citados na Seção 5.1. Outro objetivo é comparar as despesas em relação a outra plataforma de agrupamento de SCs, o SIM Array da [30], que foi introduzido na Seção 2.3.4. Portanto, cinco plataformas são comparadas.

Para calcular o custo total do SCC com cada um dos DKs, o preço dos DKs é somado ao custo dos outros dispositivos necessários ao SCC. A partir desse valor, é considerado o gasto por



(a) FIIP



(b) Controlador de Comunicação

Figura 5.1: Quantidade de elementos lógicos para cada módulo interno no FPGA.

cada *slot* de SC que a placa suporta. Além disso, é avaliada a despesa por *slot* para utilizar até 32 *slots*, com cada uma das cinco plataformas. Para as plataformas que utilizam FPGAs que não suportam essa quantidade, são necessárias placas de FPGAs adicionais. Então, é considerado o valor total com a quantidade de FPGAs suplementares utilizadas.

A Tabela 5.3 traz o custo total de implementação do SCC para cada uma das quatro placas de FPGA. Ela descreve o preço de cada um dos componentes utilizados, para o Raspberry Pi, foi obtido em [60], para o FT245 Board está em [83], e para os FPGAs estão em [82], [3], [6] e [7]. O circuito eletrônico de interface (módulo Interface, descrito na Seção 4.4) foi feito com vários dispositivos eletrônicos de prateleira, o seu desenvolvimento custou aproximadamente US$2,24. Como cada FPGA suporta uma quantidade diferente de SCs, o preço do módulo Interface é multiplicado pela quantidade de *slots* que o FPGA suporta.

O custo do SIM Array com 32 *slots*, é de US$ 9484,65, esse valor foi obtido por meio de contado direto com a empresa [30]. Na Tabela 5.4 é possível ver o preço pela quantidade de *slots*

Tabela 5.3: Custo total do SCC com diferentes placas de FPGA.

	Cyclone V DK	Stratix V DK	DE0-Nano	CoreEP4CE10
DK (US$)	1299	6995	99,95	33,99
Raspberry 3 B (US$)	42,95	42,95	42,95	42,95
FT245 Board (US$)	9,99	9,99	9,99	9,99
Circuito de Interface (US$)	71,68	56	11,2	4,48
Custo Total (US$)	1423,63	7119,62	164,09	**91,41**

suportados, para cada um dos FPGAs utilizados no SCC e para o SIM Array. A mesma tabela mostra que a placa DE0-Nano tem o menor custo por SC suportado, já o SIM Array tem o pior. O custo total do SCC é menor que o do SIM Array, para qualquer um dos FPGAs utilizados.

Tabela 5.4: Comparação do preço por *slot*, entre o SCC com quatro DKs diferentes e o SIM Array.

	SCC				SIM Array
	Cyclone V	Stratix V	DE0-Nano	CoreEP4CE10	
Qtd. de *slots* de SC	32	25	5	2	32
Custo Total (US$)	1423,63	7119,62	164,09	91,41	9484,65
Custo/*slot* (US$)	44,49	284,78	**32,82**	45,7	296,36

A Figura 5.2 demonstra o custo pela quantidade de *slots* de SCs que são utilizados. Para composição desta figura é seguido o preceito de somar o custo da quantidade de placas de FPGA necessárias para o sistema utilizar de 1 até 32 *slots* de SCs. O valor por *slot* aumenta discretamente, para quantidades de *slots* que não são múltiplas da quantidade máxima de *slots* que o FPGA suporta. Isso ocorre pois um dos FPGAs não está utilizando a capacidade máxima de *slots*. Pela mesma figura, é possível ver que a placa CoreEP4CE10 é financeiramente mais viável para aplicações com 1, 2 ou 6 SCs. Para qualquer outra quantidade a DE0-Nano é melhor. Porém, a Cyclone V DK é conveniente de ser utilizada para aplicações que utilizem de 21 a 32 *slots*, por possuir um custo por *slot* próximo das outras duas placas de baixo desempenho e precisar de apenas uma placa para gerenciar os diversos SC. Seriam necessárias 7 placas DE0-Nano ou 16 placas CoreEP4CE10 para suportar 32 SCs.

5.3 **Desempenho**

Este teste busca estudar o desempenho do SCC na transmissão de dados entre o computador hospedeiro e o SC. O SCC é avaliado até atingir o limite de SCs suportados pelo FPGA. Além disso, a escalabilidade é avaliada, ao acrescentar mais de um FPGA funcionando ao mesmo

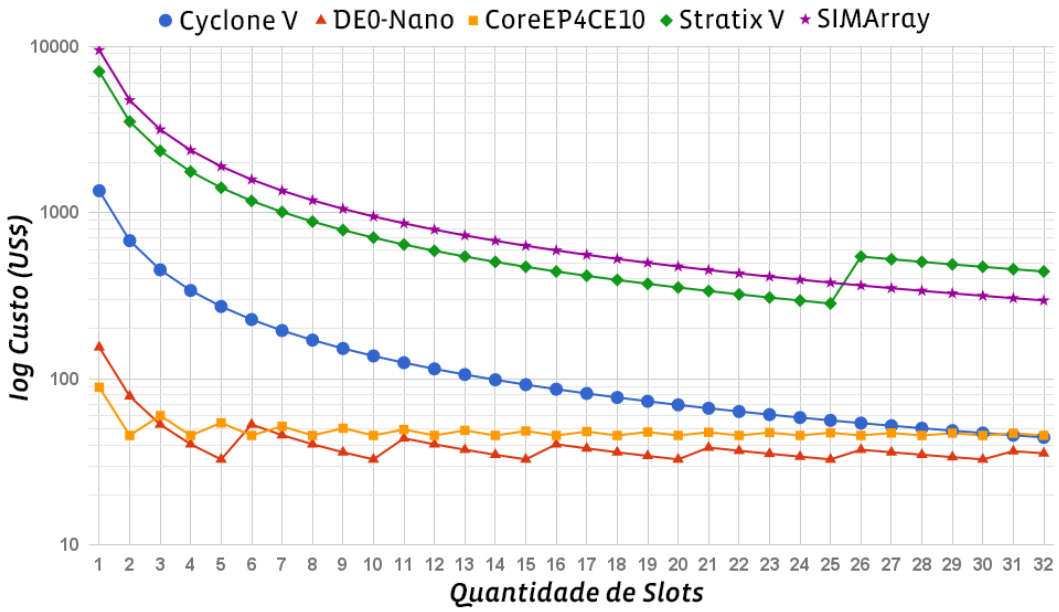


Figura 5.2: Custo pela quantidade de *slots* utilizados, para cada uma das placas de desenvolvimento aplicadas ao SCC e para o SIM Array.

tempo. Os tempos medidos são comparados com o SIM Array, que é utilizado em SESs, como foi apresentado na Seção 2.3.4. O SIM Array possui apenas um núcleo de processamento e o Driver de comunicação do SCC (Seção 4.6) só envia requisições a um SC por vez, que esteja conectado ao mesmo FPGA. No caso, ele só envia requisições ao segundo SC após o primeiro SC ter respondido. A paralelização ocorre apenas entre FPGAs diferentes.

Por meio de uma aplicação em Python, é avaliado o tempo a partir de uma requisição de reinicialização do SC, até ser recebido o ATR. O presente teste é interessante pois, na inicialização, o SC envia apenas o ATR, e ele é sempre o mesmo. Isso permite que os leitores sejam avaliados pela eficiência na transmissão de mensagens, em condições similares de execução. Os SCs utilizados são iguais, e seguem a descrição na Tabela 5.5, o ATR possui 18 Bytes.

Tabela 5.5: *Smart Card* utilizado.

Fabricante	NXP
ATR	3B E9 00 00 81 31 FE 45 4A 43 4F 50 32 31 56 32 32 A1
Tipo	Java Card v2.2 Global Platform 2.1.1
Transmissão	T=0 e T=1

São feitos quatro testes, que se diferenciam na quantidade de SCs presentes. No caso são utilizados de um até quatro SCs. Para cada teste são feitas 100 execuções, onde cada execução envia 600 requisições de ATR. O tempo total para as 600 requisições é medido. O total de requisições feitas são divididas igualmente entre os SCs presentes.

O programa de testes é executado no Raspberry pi B+, com 512MB, e processador Broadcom de 700MHz. O SCC se conecta pela interface USB, com o Raspberry. Para o SIM

Array as requisições são feitas pelo mesmo dispositivo, contudo a comunicação é TCP/IP, em rede local sem interferências, dado que ele só permite transporte de mensagens por meio desse protocolo.

O SCC será utilizado com o DK CoreEP4CE10. Como ele suporta dois *slots*, é analisado até o ponto que ele atinge a quantidade máxima de *slots*. Em seguida, é adicionado mais um FPGA, igual ao primeiro, com o objetivo de averiguar os dois FPGAs trabalhando em conjunto. Dessa maneira, será avaliado de um até o máximo de quatro *slots*.

A Tabela 5.6 e a Figura 5.3 mostram que o SCC apresenta uma significante melhoria no tempo necessário para transferir as mensagens, quando comparado ao SIM Array. O SCC processa as requisições em menor tempo, com qualquer quantidade de *slots*, e chega a ser 69% mais eficiente que o SIM Array ao utilizar 4 SCs, que são gerenciados por 2 FPGAs. Além disso, o SIM Array sempre aumenta o tempo de processamento com o aumento da quantidade SCs, chegando a ampliar em 10% o tempo de leitura com quatro SCs em relação a um SC, enquanto o SCC reduziu esse tempo em 52%. Pode ser inferido que esse aumento de tempo do SIM Array, ao acrescentar mais SCs, é devido ao processador de um núcleo e ao processo de gerenciamento de SCs implementado. Não há uma documentação detalhada dele, para serem avaliados as características de implementação que possam resultar no aumento.

Tabela 5.6: Média e desvio padrão dos tempos de 100 execuções, com 600 requisições de ATRs cada execução, para testes de 1 até 4 SCs.

	1 SC		**2 SCs**		**3 SCs**		**4 SCs**	
	μ	σ	μ	σ	μ	σ	μ	σ
SIM Array	24,00	0,005	24,92	0,044	25,67	0,298	26,42	0,481
SCC	17,03	0,042	15,99	0,038	10,74	0,032	8,14	0,032
Redução %	29%		36%		58%		**69%**	

Com o SCC, quando há o aumento de um para dois SCs, o tempo de processamento reduziu 6%. No entanto, quando é aumentado de dois para três SCs, há um redução percentual de 33%, que se deve a inclusão de um novo FPGA, o que permite a paralelização de requisições entre FPGAs e comprova a escalabilidade do SCC. Isso mostra que a utilização de uma estratégia com vários FPGAs traz um desempenho melhor que utilizar vários SCs em um mesmo FPGA. Porém, a utilização de um único FPGA traz a vantagem do agrupamento, aproveitando melhor o mesmo FPGA para a utilização de vários SCs, que podem ser utilizados para separar contextos, sem aumentar os custos.

5.4 **Estudo de caso**

O SCC é aplicado ao cenário de uma universidade pública, que foi apresentado na Seção 4.2. A estrutura de IoT possui o *middleware* IoT, que envia mensagens de controle,

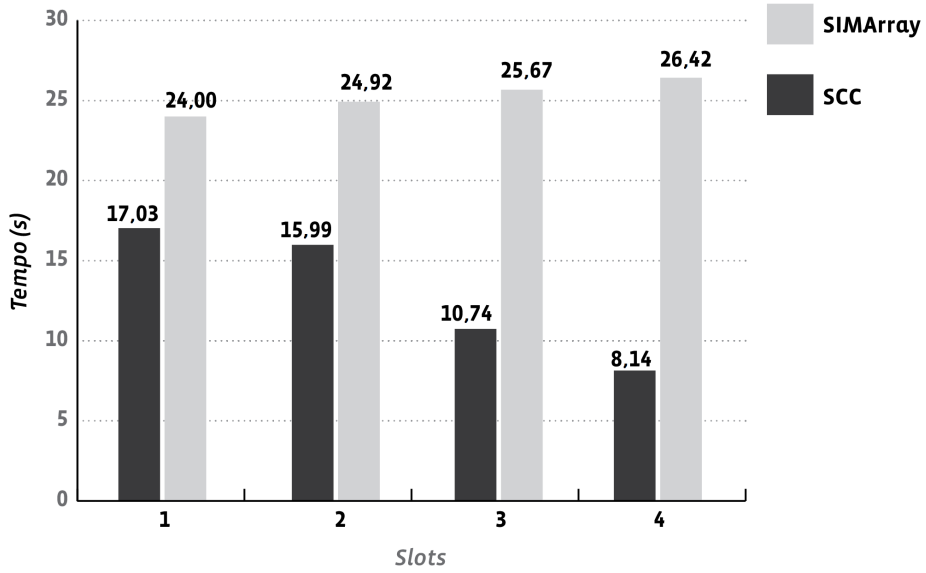


Figura 5.3: Comparação entre o SCC e o SIM Array, testando de 1 até 4 SCs e comparando as médias de 100 execuções, com 600 requisições de ATRs em cada execução.

com frequência menor que uma por minuto. Há um *nobreak* que, a cada 15 segundos, envia uma medição de potência ativa. A estrutura possui também três medidores de energia, que transmitem medições de tensão e corrente a cada 10 segundos. Além dos dispositivos enviarem suas medições, todos eles remetem o seu identificador, a unidade de medida e o marcador de data/horário que a informação foi coletada. Cada dispositivo IoT envia mensagens com formatação específica, como exemplo, a Tabela 5.7 mostra a formatação e os valores de uma mensagem real enviada pelo *nobreak*.

Tabela 5.7: Formato da mensagem e exemplo de valores enviados pelo *nobreak*.

ID	TIPO	MEDIÇÃO	UNIDADE	TEMPO
14001-Total	ACTIVE_POWER	566,4712	W	2017-05-12T17:48:32Z

Para a estrutura narrada, são aplicados dois Gateways SCC, um deles para coordenar a segurança de *nobreaks*, pois mais *nobreaks* podem ser adicionados futuramente. O outro Gateway SCC seria destinado aos medidores de energia. O Broker SCC está no *middleware* IoT. A comunicação entre duas partes é sempre feita por meio de TCP/IP, ou seja: entre dispositivos e *gateways*, entre *gateways* e *broker*, e entre *broker* e *middleware* IoT. A descrição geral dos elementos presentes na estrutura de IoT pode ser vista na Tabela 5.8, e a interação entre eles encontra-se na Figura 5.4. Na presente seção é mensurado o tempo que o SCC leva para assinar mensagens, utilizando de um até três SCs. A partir desses dados são avaliados quantos SCs o Gateway SCC e o Broker SCC devem possuir, para que suportem assinar todas as mensagens que passam por eles.

Tabela 5.8: Descrição geral dos elementos presentes na estrutura de IoT apresentada.

	Qtd	Mensagens/minuto	Mensagem	SCC
Middleware IoT	1	<1	Controle	Broker
Nobreak	1	4	Potência ativa	Gateway
Medidores de energia	3	18	Tensão e corrente	Gateway

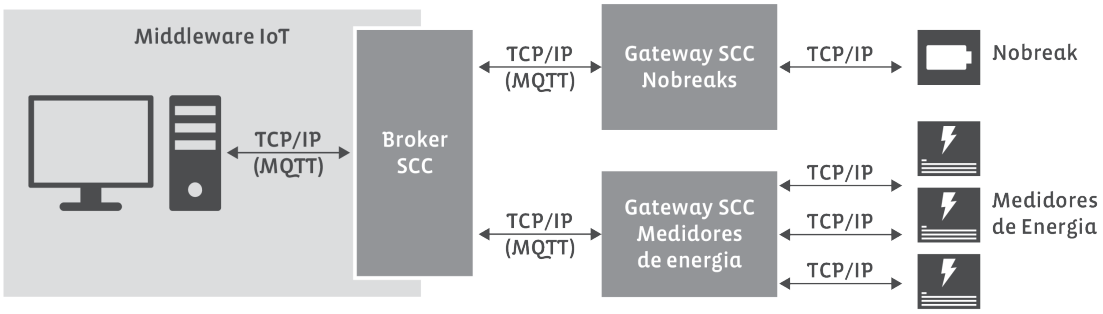


Figura 5.4: Cenário de aplicação do sistema de segurança para IoT.

Os SCs utilizados são do mesmo modelo descrito na Tabela 5.5. Para poder fazer operações de assinatura com a chave privada gerada diretamente no SC, foi instalado neles o IsoApplet [84]. Esse *applet* é empregado na criação de estruturas de dados PKCS#15 em Java Cards, permitindo utilizar a Infraestrutura de Chave Pública (*do inglês Public Key Infrastructure - PKI*). Ele assina mensagens de até 244 bytes. São geradas chaves RSA com 2048 bits, por ser o maior tamanho de chave oferecido pelos SCs utilizados e, consequentemente, ser mais segura que as chaves menores. O OpenSC é empregado para enviar as mensagens ao SC. Ele é um conjunto de bibliotecas que facilitam os processos de autenticação, criptografia e assinatura em SCs, e é compatível com o IsoApplet.

Os SCCs são estruturados com o DK CoreEP4CE10 e são utilizados como computadores hospedeiros Raspberry Pi B+, com 512MB, e processador Broadcom de 700MHz. Assim, a estrutura física do Gateway SCC e do Broker SCC, são demonstradas na Figura 5.5.

A linguagem de programação Python é utilizada para desenvolver um programa que recebe mensagens aleatórias, e transmite elas para os SCs onde serão assinadas. O tempo é medido a partir da transmissão da mensagem para o SC até o recebimento da mensagem assinada. São feitas dez execuções do programa, em cada uma delas são enviadas 60 requisições de mensagens que devem ser assinadas pelo SC. O tempo total que cada execução leva para realizar as 60 assinaturas é registrado.

Os tempos das dez execuções estão na Tabela 5.9. Com ela, é possível extrair o tempo médio para processar uma assinatura e a quantidade de assinaturas por minuto, que o SCC pode fazer com um, dois e três SCs. Estas informações estão na Tabela 5.10.

Ao aumentar de um para dois SCs num mesmo FPGA, há uma melhoria de 5% na quantidade de assinaturas processadas por minuto. Isto confirma a melhoria de desempenho,

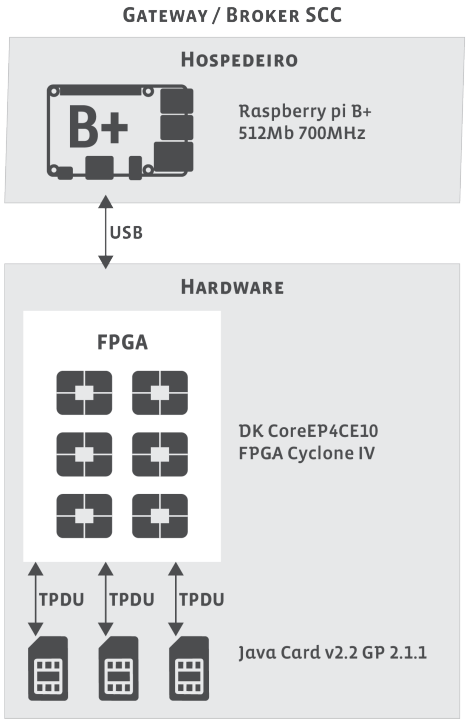


Figura 5.5: Estrutura física montada para o Gateway SCC e o Broker SCC.

Tabela 5.9: Tempo total, em segundos, para 60 requisições de assinatura em 10 execuções diferentes utilizando o SCC com 1, 2 e 3 SCs.

Execução	1 SC	2 SCs	3 SCs
1	215,59	205,03	137,63
2	215,80	205,35	137,36
3	215,66	205,61	137,21
4	215,63	205,40	138,53
5	215,43	205,64	138,07
6	215,77	205,07	137,45
7	215,63	205,03	137,81
8	215,45	205,26	137,21
9	215,46	205,33	137,08
10	215,79	205,26	136,94
Média	215,62	205,30	137,53
Desvio Padrão	0,140	0,219	0,490

Tabela 5.10: Tempo médio por assinatura, em segundos, e quantidade de assinaturas possíveis por minuto, para o SCC com 1, 2 e 3 SCs.

	1 SC	2 SCs	3 SCs
Tempo/assinatura	3,59	3,42	2,29
Assinaturas/minuto	16,70	17,54	26,18

além de trazer a vantagem de dividir contextos de processamento para SCs diferentes. Um exemplo seria utilizar SCs diferentes para processar as requisições dos medidores de energia de empresas diferentes. Porém, ao incluir um novo FPGA, com três *slots* de SCs, é possível processar 49% mais assinaturas que utilizando apenas um FPGA e dois SCs. Não há *overhead* de comunicação entre FPGAs, pois eles não são conectados entre si diretamente, ambos são gerenciados pelo computador hospedeiro. Com isso, é demonstrada a vantagem da utilização do SCC com vários FPGAs quando é desejado um melhor desempenho.

Como o Gateway SCC dos *nobreaks* recebe uma mensagem a cada 15 segundos, apenas um SC e um FPGA são suficientes para suportar a demanda de quatro mensagens por minuto. Com um SC, ele suportará até quatro estabilizadores conectados ao mesmo tempo, sem precisar aumentar a quantidade de SCs. Para o Gateway SCC dos medidores de energia, são necessários três SCs em dois FPGAs, para processar as 18 mensagens expedidas por minuto, pois cada um dos três medidores envia uma mensagem a cada dez segundos. Neste caso, dois SCs com apenas um FPGA poderiam ser utilizados, porém haveria perda de informação com mensagens eventualmente descartadas. O Gateway SCC com três SCs aceita até quatro medidores de energia, sem perder informação. O Broker SCC necessita apenas de um SC, pois ele só envia comandos de controle aos *gateways*, com frequência menor que uma mensagem por minuto. Na Figura 5.6 pode ser visto o cenário avaliado, montado com a estrutura de segurança disponibilizada pelos SCCs e a quantidade de SCs utilizados para cada um dos SCCs.

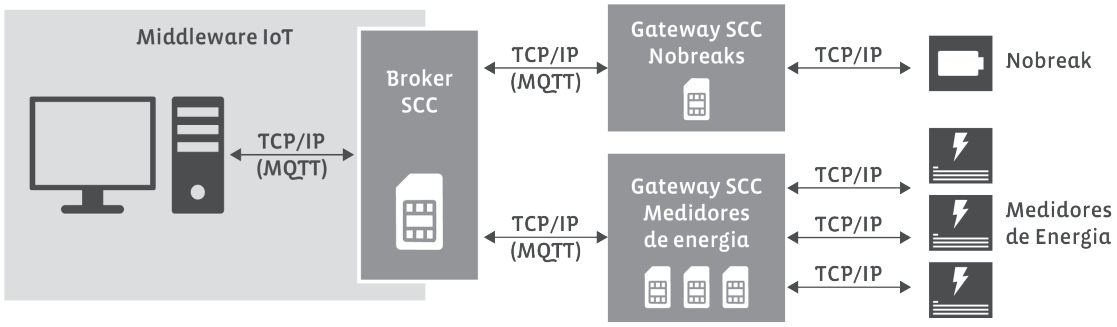


Figura 5.6: Cenário de aplicação do sistema de segurança para IoT, com a quantidade de SCs incluídos nos Gateways SCC e Broker SCC.

A Figura 5.7 mostra o processo real de transmissão de uma mensagem, a partir do *nobreak*, até ela chegar no Broker SCC. O *nobreak* começa enviando a mensagem com o formato descrito na Tabela 5.7. O Gateway SCC, através do módulo Dev Interface (Figura 4.4(a)), recebe a mensagem e a converte para a Notação de Objetos JavaScript (*do inglês JavaScript Object Notation - JSON*), resultando na mensagem descrita na Figura 5.8(a). A mensagem com nova formatação é enviada para ser assinada pelo hardware seguro, que retorna a assinatura. O Módulo P/S Gateway (Figura 4.4(a)) recebe a mensagem formatada e inclui a assinatura ao JSON, o resultado pode ser visto na Figura 5.8(b). Por fim, a mensagem é enviada para Broker SCC, a qual será verificada e repassada para o *middleware* IoT.

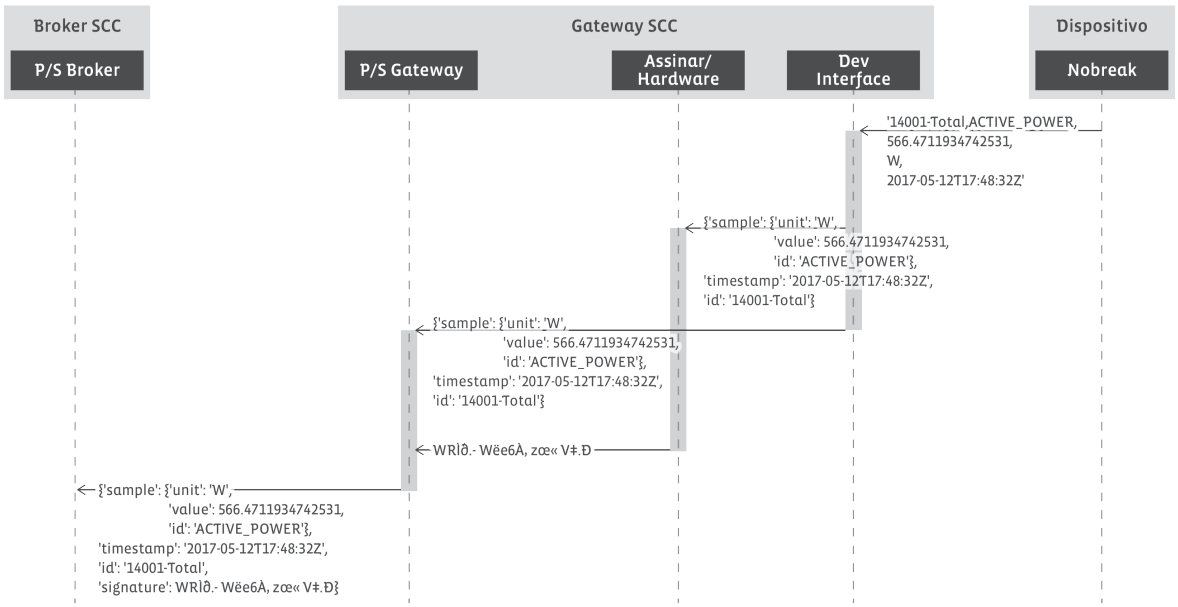


Figura 5.7: Mensagem real sendo enviada pelo dispositivo IoT, assinada pelo Gateway SCC e transmitida para o Broker SCC.

```
{'sample': {'unit': 'W',
            'value': 566.4711934742531,
            'id': 'ACTIVE_POWER'},
 'timestamp': '2017-05-12T17:48:32Z',
 'id': '14001-Total'}
```

(a) Sem assinatura

```
{'sample': {'unit': 'W',
            'value': 566.4711934742531,
            'id': 'ACTIVE_POWER'},
 'timestamp': '2017-05-12T17:48:32Z',
 'id': '14001-Total',
'signature': WRÌð.- Wëe6À, zœ« V‡.Ð}
```

(b) Com assinatura

Figura 5.8: Mensagem enviada pelo *nobreak* convertida para o formato JSON.

5.5 **Considerações finais**

Na Seção 5.1 o SCC foi aplicado a quatro FPGAs diferentes, dois de alto desempenho e dois de baixo desempenho, com o intuito de atestar a portabilidade do sistema. Todos os FPGAs suportaram o sistema, porém se diferenciam na quantidade de SCs que são capazes de gerenciar. Entre as placas testadas, as de baixo desempenho são limitadas pela quantidade de LEs, enquanto para utilizar as de alto desempenho deve ser observado o número de blocos DSP que ela oferece.

A avaliação do custo do SCC foi feita na Seção 5.2 utilizando quatro FPGAs diferentes e comparado com o SIM Array, que é uma plataforma comercial para agrupamento de SCs. Foi avaliado o custo por *slot*, levando em conta de um até 32 SCs, de forma que é considerado o valor da quantidade de FPGAs necessários para suportar o número de SCs em questão. Como o sistema é escalável permite que seja ampliado tanto a quantidade de SCs, quanto o número de FPGAs. O SIM Array possui custo por *slot* mais elevado que o SCC com qualquer uma das placas, considerando até 25 SCs. A placa DE0-Nano possui o melhor custo por *slot* para 32 SCs, mas são necessárias 4 placas para suportar essa quantidade, enquanto só é necessária uma placa Cyclone V GX para atender essa demanda.

O desempenho do sistema foi avaliado na Seção 5.3 por meio da comparação com o SIM Array. O SCC sempre melhora o seu desempenho com o aumento da quantidade de SCs, enquanto o SIM Array sempre piora. Além disso, o SCC tem um desempenho melhor que o SIM Array para qualquer uma das quantidades de SCs testadas, chegando a ser 69% mais eficiente ao utilizar 4 SCs. Nessa avaliação foi possível ampliar a quantidade de SCs e de FPGAs, o que comprova a escalabilidade do sistema. Ao ampliar de um SC para quatro SCs, o SCC melhorou em 52% o tempo de leitura.

O SCC foi aplicado a um cenário real de IoT na Seção 5.4. Ele é utilizado para assegurar a troca de mensagens entre gateway e *middleware* IoT. Foi avaliada a quantidade de assinaturas que o sistema permite por minuto, ao gerenciar um, dois e três cartões. Dessa forma, foi possível mensurar o número de SCs necessários para atender a demanda de mensagens enviadas pelos dispositivos IoT do sistema, que são um *nobreak* e três medidores de energia.

6
CONCLUSÃO

O crescente número de dispositivos conectados à IoT e a heterogeneidade da informação que circula entre eles vem tornando mais comum a ocorrência de ataques a estes sistemas, o que evidencia a necessidade por segurança. Para atender estas questões, este trabalho propôs a utilização de SCs gerenciados por FPGA, permitindo utilizar a segurança dos SCs em conjunto com a eficiência de processamento dos FPGAs. O SCC é o sistema de segurança para IoT que resulta dessa junção, o qual foi apresentado e testado neste trabalho.

A arquitetura do SCC foi exposta, mostrando a modularidade e a compatibilidade com protocolos bem estabelecidos. Comprovou-se a portabilidade do sistema ao utilizá-lo com diferentes tipos de FPGAs. Além disso, a quantidade de SCs que os mesmos FPGAs são capazes de aceitar foi mensurada, com isso foi verificado que o número de SCs para FPGAs de baixo desempenho são limitados pela quantidade de LEs, enquanto para os de alto desempenho a restrição está no número de blocos DSP.

Demonstrou-se que o SCC é mais barato que a plataforma comercial utilizada em SESs, o SIM Array. O custo por *slot* para utilizar o SCC com quatro FPGAs diferentes foi calculado considerando até 32 *slots* de SCs. Isto mostrou que as despesas de implementação do SCC podem ser adaptadas à demanda do sistema de segurança. Para utilizar 32 SCs, o sistema obteve o melhor custo benefício com placas DE0-Nano, custando US$32,82 por *slot*, enquanto o SIM Array custa US$296,36 por *slot*. A avaliação da performance do SCC foi feita por meio da comparação com o SIM Array. O SCC processa o recebimento dos bytes em menos tempo que o SIM Array para qualquer uma das quantidades de SCs testadas. Ele chegou a ser 69% mais eficiente que o SIM Array, quando os sistemas foram testado com 4 SCs. Como o SCC permitiu acrescentar mais de um SCs em um único FPGAs e também foi possível aumentar o número de FPGAs para aumentar a quantidade SCs suportados, a escalabilidade do sistema foi confirmada.

A viabilidade financeira e o desempenhos demonstrados permitem a aplicação distribuída do sistema em grupos de dispositivos, trazendo a segurança para perto deles. Isso permitiu que a plataforma fosse aplicada no contexto de uma universidade pública, proporcionando segurança às mensagens transferidas entre *gateways*, que estão em um prédio, e o *middleware* IoT, que está em outro. A proteção acontece por meio da assinatura das mensagens que oferece

integridade e autenticidade às mensagens. Ao utilizar o SCC com três SCs foi possível assinar 26,18 mensagens por minuto.

6.1 Dificuldades

Com o aumento do número de SCs no mesmo FPGA, o sistema reduz o tempo de leitura, porém não foi atingido o nível de redução esperado. Essa melhor eficiência deve ser alcançada com a evolução do sistema no FPGA para gerenciamento interno de mensagens em paralelo.

A plataforma que foi utilizada para algumas comparações, o SIM Array, possui documentação insuficiente. Além disso, o seu *driver* de comunicação com o protocolo PC/SC não é estável, impedindo a integração. Isso foi contornado através de envios diretos de mensagens através da linha de comando, utilizando o formato de mensagens definido pela empresa.

Durante os testes do SCC alguns circuitos eletrônicos de interface (Seção 4.4) apresentaram falhas, porém circuitos adicionais já tinham sido implementados, como precaução para situações dessa natureza.

6.2 Contribuições

O trabalho apresenta as seguintes contribuições:

1. Uma avaliação do consumo de recursos do sistema de gerenciamento de SCs utilizando diferentes FPGAs.
2. Uma descrição do custo pela quantidade de *slots* de SC disponíveis, levando em conta o SCC com diferentes FPGAs e comparando com uma plataforma comercial de agrupamento de SC, o SIM Array.
3. Uma análise do desempenho do SCC, mostrando sua eficiência em relação ao SIM Array e a sua escalabilidade, a qual permite aplicá-lo de forma distribuída para grupos de dispositivos IoT.
4. Uma arquitetura de segurança para IoT foi apresentada utilizando SCs gerenciados por FPGA.
5. Uma análise do SCC aplicado para assegurar uma arquitetura real de IoT.

6.3 Trabalhos Futuros

Como trabalhos futuros, planeja-se aumentar a quantidade de requisições de assinatura processadas por minuto. Com esse intuito, um *driver* de dispositivo que funcione utilizando várias *threads* deve ser implementado. Além disto, a arquitetura no FPGA deve evoluir para distribuir mensagens de forma simultânea entre os SCs. A estruturação do circuito eletrônico de

interface para funcionar com *cards* de 1,8v (Classe C), iria abranger uma maior parte dos SIM *cards* atuais.

O sistema será aplicado a cenários aos quais o reconhecimento mútuo das partes é necessário. Para isso, o sistema deverá permitir a execução de processos de autenticação e criptografia nos SCs.

Como o sistema consegue ser distribuído com baixo custo, seria apropriado utilizá-lo no contexto de Computação em Névoa (*do inglês FOG computing*), uma vez que ele se encontra mais próximo dos dispositivos na ponta da rede. Seria possível montar uma estrutura de segurança na forma de FOG, que respondesse a um servidor de segurança central na nuvem.

REFERÊNCIAS

[1] ABDMEZIEM, M. R.; TANDJAOUI, D. An end-to-end secure key management protocol for e-health applications. **Computers & Electrical Engineering**, [S.l.], v.44, p.184 – 197, 2015.

[2] ABDMEZIEM, M. R.; TANDJAOUI, D.; ROMDHANI, I. Architecting the Internet of Things: state of the art. In: ROBOTS AND SENSOR CLOUDS, Cham. **Anais...** Springer International Publishing, 2016. p.55–75.

[3] ADAFRUIT. **DE0-Nano - Altera Cyclone IV FPGA starter board**. Disponível em: `https://www.adafruit.com/product/451`, Consultado em: 20.05.17.

[4] AISSAOUI-MEHREZ, H.; URIEN, P.; PUJOLLE, G. Implementation Software to Secure Virtual Machines with Remote Grid of Secure Elements. In: MILITARY COMMUNICATIONS CONFERENCE (MILCOM), 2014 IEEE. **Anais...** [S.l.: s.n.], 2014. p.282–287.

[5] AL-FUQAHA, A. et al. Internet of things: a survey on enabling technologies, protocols, and applications. **IEEE Communications Surveys & Tutorials**, [S.l.], v.17, n.4, p.2347–2376, 2015.

[6] ALTERA. **Stratix V GX FPGA Development Kit**. Disponível em: `https://www.altera.com/products/boards_and_kits/dev-kits/altera/kit-sv-gx-host.html`, Consultado em: 20.05.17.

[7] ALTERA. **Cyclone V GT FPGA Development Kit**. Disponível em: `https://www.altera.com/products/boards_and_kits/dev-kits/altera/kit-cyclone-v-gt.html`, Consultado em: 20.05.17.

[8] ALTERA. **Quartus II Subscription Edition**. Disponível em: `http://dl.altera.com/13.1/`, Consultado em: 20.05.17.

[9] ALTERA. **How do I interpret the Logic Utilization number reported in the Quartus II Fitter report?** Disponível em: `https://www.altera.com/support/support-resources/knowledge-base/solutions/rd05172012_146.html`, Consultado em: 20.05.17.

[10] AMAN, M. N.; CHUA, K. C.; SIKDAR, B. Secure Data Provenance for the Internet of Things. In: ACM INTERNATIONAL WORKSHOP ON IOT PRIVACY, TRUST, AND SECURITY, 3., New York, NY, USA. **Proceedings...** ACM, 2017. p.11–14. (IoTPTS '17).

[11] ANIRUDH, M.; THILEEBAN, S. A.; NALLATHAMBI, D. J. Use of honeypots for mitigating DoS attacks targeted on IoT networks. In: INTERNATIONAL CONFERENCE ON COMPUTER, COMMUNICATION AND SIGNAL PROCESSING (ICCCSP), 2017. **Anais...** [S.l.: s.n.], 2017. p.1–4.

[12] ATALLAH, E. et al. **A Grid of Java Cards to Deal with Security Demanding Application Domains**. 2005.

[13] BOKEFODE, J. D. et al. Developing A Secure Cloud Storage System for Storing IoT Data by Applying Role Based Encryption. **Procedia Computer Science**, [S.l.], v.89, p.43 – 50, 2016. Twelfth International Conference on Communication Networks, {ICCN} 2016, August 19– 21, 2016, Bangalore, India Twelfth International Conference on Data Mining and Warehousing, {ICDMW} 2016, August 19-21, 2016, Bangalore, India Twelfth International Conference on Image and Signal Processing, {ICISP} 2016, August 19-21, 2016, Bangalore, India.

[14] CHAUMETTE, S. et al. Secure distributed computing on a Java Card™ grid. In: IEEE INTERNATIONAL PARALLEL AND DISTRIBUTED PROCESSING SYMPOSIUM, 19. **Anais...** [S.l.: s.n.], 2005. p.8–pp.

[15] CHAUMETTE, S.; KARRAY, A.; SAUVERON, D. Secure collaborative and distributed services in the java card grid platform. In: INTERNATIONAL SYMPOSIUM ON COLLABORATIVE TECHNOLOGIES AND SYSTEMS (CTS'06). **Anais...** [S.l.: s.n.], 2006. p.56–63.

[16] CHEN, H.-C. et al. A security gateway application for End-to-End {M2M} communications. **Computer Standards & Interfaces**, [S.l.], v.44, p.85 – 93, 2016.

[17] CHEN, Y.-W. et al. Group-based authentication and key agreement. **Wireless Personal Communications**, [S.l.], v.62, n.4, p.965–979, 2010.

[18] COOLRUNNER-II Smart Card Reader. [S.l.]: Xilinx, 2003. Disponível em: `http://www.xilinx.com/support`, Consultado em: 20.05.17.

[19] DHILLON, P. K.; KALRA, S. A lightweight biometrics based remote user authentication scheme for IoT services. **Journal of Information Security and Applications**, [S.l.], p.–, 2017.

[20] DJEDJIG, N. et al. New trust metric for the RPL routing protocol. In: INTERNATIONAL CONFERENCE ON INFORMATION AND COMMUNICATION SYSTEMS (ICICS), 2017. **Anais...** [S.l.: s.n.], 2017. p.328–335.

[21] EVANS, D. The internet of things: how the next evolution of the internet is changing everything. **CISCO white paper**, [S.l.], v.1, n.2011, p.1–11, 2011.

[22] FOX-BREWSTER, T. **How Hacked Cameras Are Helping Launch The Biggest Attacks The Internet Has Ever Seen**. Disponível em: `https://www.forbes.com/sites/thomasbrewster/2016/09/25/brian-krebs-overwatch-ovh-smashed-by-largest-ddos-attacks-ever/`, Consultado em: 20.05.17.

[23] FREET, D.; AGRAWAL, R. An Overview of Architectural and Security Considerations for Named Data Networking (NDN). In: INTERNATIONAL CONFERENCE ON MANAGEMENT OF DIGITAL ECOSYSTEMS, 8., New York, NY, USA. **Proceedings...** ACM, 2016. p.52–57. (MEDES).

[24] FT245. **FT245R USB FIFO IC**. [S.l.]: Future Technology Devices International Ltd., 2010. Version 2.13.

[25] GARTNER. Gartner Says 8.4 Billion Connected "Things"Will Be in Use in 2017, Up 31 Percent From 2016. **Gart. Inc**, [S.l.], 2016. Disponível em: http://www.gartner.com/newsroom/id/3598917, Consultado em: 20.05.17.

[26] GREENBERG, A. **Hackers Remotely Kill a Jeep on the Highway—With Me in It**. https://www.wired.com/2015/07/hackers-remotely-kill-jeep-highway/.

[27] HAFEEZ, I. et al. Securebox: toward safer and smarter iot networks. In: ACM WORKSHOP ON CLOUD-ASSISTED NETWORKING, 2016., New York, NY, USA. **Proceedings...** ACM, 2016. p.55–60. (CAN '16).

[28] HANNA, S. **Hardware Is the Foundation of IoT Security**. Disponível em: https://www.globalsign.com/en/blog/iot-security-hardware/, Consultado em: 20.05.17.

[29] HO, G. et al. Smart Locks: lessons for securing commodity internet of things devices. In: ACM ON ASIA CONFERENCE ON COMPUTER AND COMMUNICATIONS SECURITY, 11., New York, NY, USA. **Proceedings...** ACM, 2016. p.461–472. (ASIA CCS '16).

[30] IMPLEMENTA. **implementa.com, SIM Array**. Disponível em: http://www.implementa.com/company/contact.html, Consultado em: 20.05.17.

[31] ISO/IEC 7810 . **ISO/IEC 7810**:2003 - identification cards – physical characteristics. 2003.

[32] ISO/IEC 7816-3. **Cards with contacts — Electrical interface and transmission protocols**. Disponível em: https://www.iso.org/obp/ui/#iso:std:iso-iec:7816:-3:en, Consultado em: 20.05.17.

[33] ISO/IEC 7816-4. **Organization, security and commands for interchange**. Disponível em: https://www.iso.org/obp/ui/#iso:std:iso-iec:7816:-4:en, Consultado em: 20.05.17.

[34] JAYARAMAN, P. P. et al. Privacy preserving Internet of Things: from privacy techniques to a blueprint architecture and efficient implementation. **Future Generation Computer Systems**, [S.l.], p.–, 2017.

[35] JERALD, A. V.; RABARA, S. A.; BAI, D. P. Secure IoT architecture for integrated smart services environment. In: INTERNATIONAL CONFERENCE ON COMPUTING FOR SUSTAINABLE GLOBAL DEVELOPMENT (INDIACOM), 2016. **Anais...** [S.l.: s.n.], 2016. p.800–805.

[36] JIANG, H. et al. A secure and scalable storage system for aggregate data in IoT. **Future Generation Computer Systems**, [S.l.], v.49, p.133 – 141, 2015.

[37] KAHVAZADEH, S. et al. Securing Combined Fog-to-Cloud System Through SDN Approach. In: WORKSHOP ON CROSSCLOUD INFRASTRUCTURES & PLATFORMS, 4., New York, NY, USA. **Proceedings...** ACM, 2017. p.2:1–2:6. (Crosscloud'17).

[38] KARIMIAN, N.; WORTMAN, P. A.; TEHRANIPOOR, F. Evolving Authentication Design Considerations for the Internet of Biometric Things (IoBT). In: ELEVENTH IEEE/ACM/IFIP INTERNATIONAL CONFERENCE ON HARDWARE/SOFTWARE CODESIGN AND SYSTEM SYNTHESIS, New York, NY, USA. **Proceedings...** ACM, 2016. p.10:1–10:10. (CODES '16).

[39] KIM, Y. S.; TAGUE, P. Proximity-based Wireless Access Control Through Considerate Jamming. In: ACM MOBICOM WORKSHOP ON SECURITY AND PRIVACY IN MOBILE ENVIRONMENTS, New York, NY, USA. **Proceedings...** ACM, 2014. p.19–24. (SPME '14).

[40] KO, H.; JIN, J.; KEOH, S. L. Secure Service Virtualization in IoT by Dynamic Service Dependency Verification. **IEEE Internet of Things Journal**, [S.l.], v.3, n.6, p.1006–1014, Dec 2016.

[41] KOUL, Y.; PATHAK, A. Design and FPGA-based implementation of Smartcard Reader. **International Journal of Science, Engineering and Technology Research (IJSETR)**, [S.l.], v.5, February 2016.

[42] LEO, M. et al. A federated architecture approach for Internet of Things security. In: EURO MED TELCO CONFERENCE (EMTC), 2014. **Anais...** [S.l.: s.n.], 2014. p.1–5.

[43] LEVI, A.; SARIMURAT, S. Utilizing hash graphs for key distribution for mobile and replaceable interconnected sensors in the IoT context. **Ad Hoc Networks**, [S.l.], v.57, p.3 – 18, 2017. Special Issue on Internet of Things and Smart Cities security, privacy and new technologies.

[44] MAHMOOD, K. et al. A lightweight message authentication scheme for Smart Grid communications in power sector. **Computers & Electrical Engineering**, [S.l.], v.52, p.114 – 124, 2016.

[45] MAHMOUD, R. et al. Internet of things (IoT) security: current status, challenges and prospective measures. In: INTERNET TECHNOLOGY AND SECURED TRANSACTIONS (ICITST), 2015 10TH INTERNATIONAL CONFERENCE FOR. **Anais...** [S.l.: s.n.], 2015. p.336–341.

[46] MäKI, P. et al. Interface Diversification in IoT Operating Systems. In: INTERNATIONAL CONFERENCE ON UTILITY AND CLOUD COMPUTING, 9., New York, NY, USA. **Proceedings...** ACM, 2016. p.304–309. (UCC '16).

[47] MANDI, B. C.; VENKATESWARAN, P.; NANDI, R. FPGA based Smart Card reader. In: INTERNATIONAL CONFERENCE ON COMMUNICATION AND INDUSTRIAL APPLICATION, 2011. **Anais...** [S.l.: s.n.], 2011. p.1–4.

[48] MAXFIELD, C. **The design warrior's guide to FPGAs**: devices, tools and flows. [S.l.]: Elsevier, 2004.

[49] MEHMOOD, R. et al. IoT-enabled Web Warehouse Architecture: a secure approach. **Personal Ubiquitous Comput.**, London, UK, UK, v.19, n.7, p.1157–1167, Oct. 2015.

[50] MISRA, G. et al. Internet of things (iot)–a technological analysis and survey on vision, concepts, challenges, innovation directions, technologies, and applications (an upcoming or

future generation computer communication system technology). **American Journal of Electrical and Electronic Engineering**, [S.l.], v.4, n.1, p.23–32, 2016.

[51] MOOSAVI, S. R. et al. An Elliptic Curve-based Mutual Authentication Scheme for {RFID} Implant Systems. **Procedia Computer Science**, [S.l.], v.32, p.198 – 206, 2014. The 5th International Conference on Ambient Systems, Networks and Technologies (ANT-2014), the 4th International Conference on Sustainable Energy Information Technology (SEIT-2014).

[52] MORATELLI, C. et al. Embedded Virtualization for the Design of Secure IoT Applications. In: INTERNATIONAL SYMPOSIUM ON RAPID SYSTEM PROTOTYPING: SHORTENING THE PATH FROM SPECIFICATION TO PROTOTYPE, 27., New York, NY, USA. **Proceedings...** ACM, 2016. p.2–6. (RSP '16).

[53] NAWIR, M. et al. Internet of Things (IoT): taxonomy of security attacks. In: ELECTRONIC DESIGN (ICED), 2016 3RD INTERNATIONAL CONFERENCE ON. **Anais...** [S.l.: s.n.], 2016. p.321–326.

[54] NGUYEN, H. V.; IACONO, L. L. REST-ful CoAP Message Authentication. In: INTERNATIONAL WORKSHOP ON SECURE INTERNET OF THINGS (SIOT), 2015. **Anais...** [S.l.: s.n.], 2015. p.35–43.

[55] NTULI, N.; ABU-MAHFOUZ, A. A Simple Security Architecture for Smart Water Management System. **Procedia Computer Science**, [S.l.], v.83, p.1164 – 1169, 2016. The 7th International Conference on Ambient Systems, Networks and Technologies (ANT 2016) / The 6th International Conference on Sustainable Energy Information Technology (SEIT-2016) / Affiliated Workshops.

[56] OLIVIER, F.; CARLOS, G.; FLORENT, N. New Security Architecture for IoT Network. **Procedia Computer Science**, [S.l.], v.52, p.1028 – 1033, 2015. The 6th International Conference on Ambient Systems, Networks and Technologies (ANT-2015), the 5th International Conference on Sustainable Energy Information Technology (SEIT-2015).

[57] OWENS, C. **Stranger hacks family's baby monitor and talks to child at night**. Disponível em: `http://sfglobe.com/2016/01/06/stranger-hacks-familys-baby-monitor-and-talks-to-child-at-night/`, Consultado em: 20.05.17.

[58] PC/SC Specification. Disponível em: `https://www.pcscworkgroup.com/`, Consultado em: 20.05.17.

[59] RANKL, W.; EFFING, W. **Smart Card Handbook**. 4th.ed. Germany: Wiley, 2010.

[60] RS COMPONENTS LTD. **Raspberry Pi 3 Model B SBC**. Disponível em: `http://uk.rs-online.com/web/p/processor-microcontroller-development-kits/8968660/`, Consultado em: 20.05.17.

[61] SAEED, A. et al. Intelligent Intrusion Detection in Low-Power IoTs. **ACM Trans. Internet Technol.**, New York, NY, USA, v.16, n.4, p.27:1–27:25, Dec. 2016.

[62] SCIANCALEPORE, S. et al. Key Management Protocol with Implicit Certificates for IoT Systems. In: WORKSHOP ON IOT CHALLENGES IN MOBILE AND INDUSTRIAL

SYSTEMS, 2015., New York, NY, USA. **Proceedings...** ACM, 2015. p.37–42. (IoT-Sys '15).

[63] SCIANCALEPORE, S. et al. LICITUS: a lightweight and standard compatible framework for securing layer-2 communications in the iot. **Computer Networks**, [S.l.], v.108, p.66 – 77, 2016.

[64] SEMICONDUCTORS, N. **Level shifting techniques in I2C-bus design, AN10441, Rev. 01**. Disponível em: `http://www.nxp.com/documents/application_note/AN10441.pdf`, Consultado em: 20.05.17.

[65] SHAH, D.; HARADI, V. IoT Based Biometrics Implementation on Raspberry Pi. **Procedia Computer Science**, [S.l.], v.79, p.328 – 336, 2016. Proceedings of International Conference on Communication, Computing and Virtualization (ICCCV) 2016.

[66] SINGH, A. et al. Exploring power attack protection of resource constrained encryption engines using integrated low-drop-out regulators. In: IEEE/ACM INTERNATIONAL SYMPOSIUM ON LOW POWER ELECTRONICS AND DESIGN (ISLPED), 2015. **Anais...** [S.l.: s.n.], 2015. p.134–139.

[67] SKERRETT, I. **IoT Developer Survey 2017**. Disponível em: `https://ianskerrett.wordpress.com/2017/04/19/iot-developer-trends-2017-edition/`, Consultado em: 20.05.17.

[68] STEWART, C. E.; VASU, A. M.; KELLER, E. CommunityGuard: a crowdsourced home cyber-security system. In: ACM INTERNATIONAL WORKSHOP ON SECURITY IN SOFTWARE DEFINED NETWORKS & NETWORK FUNCTION VIRTUALIZATION, New York, NY, USA. **Proceedings...** ACM, 2017. p.1–6. (SDN-NFVSec '17).

[69] SU, W.-T.; WONG, W.-M.; CHEN, W.-C. A survey of performance improvement by group-based authentication in IoT. In: APPLIED SYSTEM INNOVATION (ICASI), 2016 INTERNATIONAL CONFERENCE ON. **Anais...** [S.l.: s.n.], 2016. p.1–4.

[70] TAHIR, H.; TAHIR, R.; MCDONALD-MAIER, K. Securing MEMS Based Sensor Nodes in the Internet of Things. In: SIXTH INTERNATIONAL CONFERENCE ON EMERGING SECURITY TECHNOLOGIES (EST), 2015. **Anais...** [S.l.: s.n.], 2015. p.44–49.

[71] TAO, M. et al. Multi-layer cloud architectural model and ontology-based security service framework for IoT-based smart homes. **Future Generation Computer Systems**, [S.l.], p.–, 2016.

[72] TAO, S.; DUBROVA, E. Reliable Low-overhead Arbiter-based Physical Unclonable Functions for Resource-constrained IoT Devices. In: FOURTH WORKSHOP ON CRYPTOGRAPHY AND SECURITY IN COMPUTING SYSTEMS, New York, NY, USA. **Proceedings...** ACM, 2017. p.1–6. (CS2 '17).

[73] TRAMPUS, M. et al. Using smart cards as a secure storage for digitally signed documents. In: THE IEEE REGION 8 EUROCON 2003. COMPUTER AS A TOOL. **Anais...** [S.l.: s.n.], 2003. v.2, p.74–78 vol.2.

[74] TRIMBERGER, S. M. Three ages of FPGAs: a retrospective on the first thirty years of fpga technology. **Proceedings of the IEEE**, [S.l.], v.103, n.3, p.318–331, 2015.

[75] UKIL, A. et al. Lightweight Security Scheme for Vehicle Tracking System Using CoAP. In: INTERNATIONAL WORKSHOP ON ADAPTIVE SECURITY, New York, NY, USA. **Proceedings...** ACM, 2013. p.3:1–3:8. (ASPI '13).

[76] URIEN, P. Cloud of secure elements perspectives for mobile and cloud applications security. In: COMMUNICATIONS AND NETWORK SECURITY (CNS), 2013 IEEE CONFERENCE ON. **Anais...** [S.l.: s.n.], 2013. p.371–372.

[77] URIEN, P. **Cloud of Secure Elements(CoSE), draft-urien-cfrg-cose-00**. 2014.

[78] URIEN, P. Cloud of secure elements: an infrastructure for the trust of mobile nfc services. In: WIRELESS AND MOBILE COMPUTING, NETWORKING AND COMMUNICATIONS (WIMOB), 2014 IEEE 10TH INTERNATIONAL CONFERENCE ON. **Anais...** [S.l.: s.n.], 2014. p.213–218.

[79] URIEN, P.; MARIE, E.; KIENNERT, C. An innovative solution for cloud computing authentication: grids of eap-tls smart cards. In: DIGITAL TELECOMMUNICATIONS (ICDT), FIFTH INTERNATIONAL CONFERENCE ON. **Anais...** [S.l.: s.n.], 2010. p.22–27.

[80] URIEN, P.; PIRAMUTHU, S. Towards a Secure Cloud of Secure Elements Concepts and Experiments with NFC Mobiles. In: CTS 2013 CONFERENCE. **Anais...** May, 2013.

[81] VARAKLIOTIS, S.; KIRSTEIN, P. T.; DEIANA, G. The use of Handle to aid IoT security. In: IEEE INTERNATIONAL CONFERENCE ON COMMUNICATIONS (ICC), 2015. **Anais...** [S.l.: s.n.], 2015. p.542–548.

[82] WAVESHARE. **OpenEP4CE10-C, ALTERA Development Board**. Disponível em: `http://www.waveshare.com/coreep4ce10.htm`, Consultado em: 20.05.17.

[83] WAVESHARE. **FT245 USB FIFO Board (mini)**. Disponível em: `http://www.waveshare.com/product/FT245-USB-FIFO-Board-mini.htm`, Consultado em: 20.05.17.

[84] WENDLAND, P. **IsoApplet - A Java Card PKI Applet aiming to be ISO 7816 compliant**. Disponível em: `https://github.com/philipWendland/IsoApplet`, Consultado em: 07.06.17.

[85] WILLERS, O. et al. MEMS Gyroscopes As Physical Unclonable Functions. In: ACM SIGSAC CONFERENCE ON COMPUTER AND COMMUNICATIONS SECURITY, 2016., New York, NY, USA. **Proceedings...** ACM, 2016. p.591–602. (CCS '16).

[86] WU, M. et al. Research on the architecture of Internet of Things. In: INTERNATIONAL CONFERENCE ON ADVANCED COMPUTER THEORY AND ENGINEERING(ICACTE), 2010. **Anais...** [S.l.: s.n.], 2010. v.5, p.V5–484–V5–487.

[87] XU, H. et al. Efficient P2P-based mutual authentication protocol for {RFID} system security of {EPC} network using asymmetric encryption algorithm. **The Journal of China Universities of Posts and Telecommunications**, [S.l.], v.18, Supplement 1, p.40 – 47, 2011.

[88] YANG, J. cui; PANG, H.; ZHANG, X. Enhanced mutual authentication model of IoT. **The Journal of China Universities of Posts and Telecommunications**, [S.l.], v.20, Supplement 2, p.69 – 74, 2013.

[89] YANG, K.; FORTE, D.; TEHRANIPOOR, M. M. Protecting Endpoint Devices in IoT Supply Chain. In: IEEE/ACM INTERNATIONAL CONFERENCE ON COMPUTER-AIDED DESIGN, Piscataway, NJ, USA. **Proceedings...** IEEE Press, 2015. p.351–356. (ICCAD '15).

[90] YANG, K.; FORTE, D.; TEHRANIPOOR, M. M. CDTA: a comprehensive solution for counterfeit detection, traceability, and authentication in the iot supply chain. **ACM Trans. Des. Autom. Electron. Syst.**, New York, NY, USA, v.22, n.3, p.42:1–42:31, Apr. 2017.

[91] YU, W.; KöSE, S. A Lightweight Masked AES Implementation for Securing IoT Against CPA Attacks. **IEEE Transactions on Circuits and Systems I: Regular Papers**, [S.l.], v.PP, n.99, p.1–11, 2017.

[92] ZENGER, C. T. et al. Exploiting the Physical Environment for Securing the Internet of Things. In: NEW SECURITY PARADIGMS WORKSHOP, 2015., New York, NY, USA. **Proceedings...** ACM, 2015. p.44–58. (NSPW '15).

[93] ZHANG, K. et al. Security and Privacy in Smart City Applications: challenges and solutions. **IEEE Communications Magazine**, [S.l.], v.55, n.1, p.122–129, January 2017.

[94] ZHANG, Z.-K. et al. IoT security: ongoing challenges and research opportunities. In: SERVICE-ORIENTED COMPUTING AND APPLICATIONS (SOCA), 2014 IEEE 7TH INTERNATIONAL CONFERENCE ON. **Anais...** [S.l.: s.n.], 2014. p.230–234.

[95] ZHAO, K.; GE, L. A Survey on the Internet of Things Security. In: NINTH INTERNATIONAL CONFERENCE ON COMPUTATIONAL INTELLIGENCE AND SECURITY, 2013. **Anais...** [S.l.: s.n.], 2013. p.663–667.

[96] ZUO, C. et al. CCA-secure {ABE} with outsourced decryption for fog computing. **Future Generation Computer Systems**, [S.l.], p.–, 2016.