Esquema de Autenticação e Acordo de Chaves para Internet das Coisas

Degemar Pereira da Silva[1], **Ramon Fontes**[1], **Augusto Neto**[2], **Gustavo Girao Barreto Da Silva**[1], **Roger Immich**[1]

[1] Instituto Metrópole Digital (IMD)
Universidade Federal do Rio Grande do Norte (UFRN)

[2]Departamento de Informática e Matemática Aplicada (DIMAp)
Universidade Federal do Rio Grande do Norte (UFRN)

degemar.pereira@ufrn.edu.br, augusto@dimap.ufrn.br,
{ramon.fontes, girao, roger}@imd.ufrn.br

Abstract. *The Internet of Things (IoT) continues to advance by leaps and bounds, allowing more and more devices to be connected to the Internet every day. This technology faces several challenges, including information security and privacy. In IoT environments, security is essential to prevent the entry of malicious devices, provide secure communication, and protect sensitive data. The present work presents* ***AuThenTication*** *and Key* ***A****greement* ***sCHeme*** *for Internet of Things (ATTACH-IoT). The proposed scheme was developed using XOR logic gates, symmetric cryptography, and hash functions. In addition, the Physical Unclonable Functions (PUF) technique was adopted for the unique and automatic identification of IoT devices, thus allowing the environment to be configured autonomously without needing to add a user and password for each of the devices. A proof of concept was implemented using containers, and formal validation was performed using Scyther. The results showed the proposed scheme's efficiency, meeting all the security requirements tested by the tool.*

Resumo. *A Internet das Coisas (IoT) continua avançando a passos largos, permitindo que a cada dia mais dispositivos estejam conectados na Internet. Esta tecnologia enfrenta diversos desafios, sendo que um dos principais é a segurança da informação. Em ambientes IoT a segurança é essencial para impedir a entrada de dispositivos mal intencionados, prover comunicação segura e proteção a dados sigilosos. O presente trabalho apresenta um esquema de* ***AuTenTicação*** *e* ***A****cordo de* ***CH****aves para* ***IoT*** *(ATTACH-IoT). O esquema proposto foi desenvolvido utilizando portas lógicas XOR, criptografia simétrica e funções hash. Complementarmente, foi adotada a técnica de Physical Unclonable Functions (PUF) para a identificação única e automática de dispositivos em IoT, permitindo assim que o ambiente seja configurado de forma autônoma, sem a necessidade de adicionar usuário e senha em cada um dos dispositivos. Foi implementada uma prova de conceito utilizando contêineres e realizada a validação formal através do Scyther. Os resultados mostraram a eficiência do esquema proposto, atendendo a todos os requisitos de segurança testados pela ferramenta.*

1. Introdução

A Internet das Coisas (*Internet of Things* - IoT) é um conceito que tem ganhado uma grande notoriedade nos últimos anos. Essa tecnologia tem possibilitado o desenvolvimento de novos dispositivos inteligentes e inovadores, permitindo que diferentes tipos de aparelhos possam se comunicar utilizando rede sem fio [Bittencourt et al. 2018]. Com o aumento do número destes dispositivos, o volume de dados em tráfego nas redes passou a ter um aumento acelerado [do Prado et al. 2021, Fiorenza and et al. 2021]. Com isto, surge um problema, que é a segurança desses elementos [Kreutz et al. 2020, Fernandes et al. 2020]. Atualmente, diversos ambientes IoT, à medida que vão se expandindo, também vão necessitando de sistemas de segurança melhores [Hassija et al. 2019]. Como resultado disso, existe a necessidade da criação de novos algoritmos de segurança que geralmente exigem recursos de hardware robustos o suficiente para serem executados, o que contrapõe com as limitações de hardware dos dispositivos IoT em geral.

Algumas propriedades da segurança da informação podem ser levadas em consideração quando tratamos de segurança para dispositivos IoT [Garcia-Morchon et al. 2019], a exemplo da proteção de dados, comunicação segura, autenticação e identificação. Outra dificuldade latente em IoT é a identificação dos dispositivos. Para esta finalidade, é possível a utilização da técnica de PUF para prover identificação única dos dispositivos. PUF é uma técnica de segurança que utiliza a microestrutura física do dispositivo que ocorre durante a sua fabricação para gerar o seu identificador único [Rührmair and Holcomb 2014]. Essa microestrutura depende de fatores físicos e randômicos que são imprevisíveis durante a construção do dispositivo. Sendo assim, é virtualmente impossível que dois chips fabricados sejam exatamente iguais, o que indica que a técnica de PUF possui boa resistência a ataques que buscam clonar a identidade do dispositivo [Zhu et al. 2019].

Com o objetivo de avançar o estado da arte, propomos um esquema de **A**u**T**en**T**icação e **A**cordo de **CH**aves para **IoT** (ATTACH-IoT). O esquema proposto é capaz de ser utilizado em conjunto com qualquer protocolo de comunicação para IoT. A proposta apresenta um conjunto de algoritmos leves que visam consumo reduzido de recursos computacionais, característica imperativa em um contexto onde grande parte dos dispositivos utilizados são caracterizados por possuírem recursos restritos e baixo poder energético [Pisani et al. 2020]. Além disso, o esquema proposto não requer a configuração individual, pois através da técnica de PUF é possível gerar uma identidade única, que é utilizada no processo de autenticação e acordo de chaves.

O ATTACH-IoT foi implementado em um ambiente virtual utilizando contêineres, realizando todo o processo de autenticação dos clientes e publicação de mensagens *Message Queuing Telemetry Transport* (MQTT) após autorização do servidor. O esquema também foi validado formalmente utilizando o *Scyther* [Cremers 2008]. Ao final, o esquema proposto foi capaz de atender aos requisitos de segurança, como por exemplo, proteção de dados, ataques do tipo *man-in-the-middle* e anonimidade do usuário.

O restante deste trabalho foi dividido da seguinte forma. A Seção 2 apresenta os trabalhos relacionados; a arquitetura proposta é apresentada na Seção 3; as Seções 4 e 5 descrevem como foi realizada a implementação e a avaliação, respectivamente. Por fim, a Seção 6 finaliza este trabalho com as considerações finais.

2. Trabalhos Relacionados

Em [il Bae and Kwak 2017] foi desenvolvido um trabalho buscando trazer um novo algoritmo de autenticação para IoT. Essa solução consiste em um protocolo de autenticação baseado em *smart-card*, que realiza o processo de autenticação utilizando um cartão inteligente. Tanto clientes como o servidor da rede IoT utilizam este cartão inteligente para se autenticarem. A proposta apresentou bons resultados, mitigando diversos tipos de ataques. Contudo, a solução proposta ainda sofre com algumas ameaças, a exemplo de não prover anonimidade do usuário e autenticação mútua segura.

Uma solução de autenticação mútua baseada em três fatores foi apresentada por [Lee et al. 2019]. O protocolo proposto utiliza um elemento que autentica tanto o *gateway* da rede quanto os usuários. Os testes realizados mostraram que a solução mitiga alguns problemas de segurança. Como desvantagem, a solução acaba dependendo inteiramente de uma entidade para autenticar todas as entidades da rede, criando um ponto único de falha. Caso este falhe, a segurança da rede fica comprometida.

Em [Zhu et al. 2019] foi desenvolvido um protocolo de autenticação mútua para RFID utilizando técnicas de PUF. É utilizado como método principal de identificação tags RFID e funções PUF. Além disso, foram utilizados métodos como portas lógicas XOR e funções hash. Os testes de segurança realizados no *Scyther* mostraram eficiência do esquema, protegendo contra ataques. Apesar de ser uma solução escalável, ela necessita de uma grande taxa de armazenamento para os dados gerados durante a autenticação.

Um protocolo de autenticação para IoT voltado para casas inteligentes foi proposto em [Oh et al. 2021]. O protocolo é baseado em quatro etapas: inicialização, registro, autenticação e acordo de chaves e atualização de chaves. Para isso, foram utilizadas técnicas como lógicas XOR e funções hash. A proposta conseguiu mitigar os tipos de ataques analisados. Como ponto negativo, a solução depende de um dispositivo que atua como um registrador dentro da rede, em consequência disso, o processo de registro dos dispositivos fica inteiramente responsável por ele, que caso pare de funcionar, a segurança da rede fica comprometida. A Tabela 1 mostra de forma resumida as características dos trabalhos.

Tabela 1. Comparações com os Trabalhos Relacionados

Trabalho	Técnicas Usadas	Vantagens	Desvantagens
[il Bae and Kwak 2017]	Cartões inteligentes para autenticação	Proteção contra ataques de negação de serviço e do tipo replay	Não prover anonimidade do usuário; autenticação mútua segura
[Lee et al. 2019]	Autenticador central; Cartões inteligentes	Baixo custo de comunicação	Dependência de um elemento central para fazer todas as autenticações
[Zhu et al. 2019]	Funções hash; Operações XOR	Identificação única usando técnica PUF	Necessário alto espaço de armazenamento para os dados gerados
[Oh et al. 2021]	Funções hash; Operações XOR	Baixo custo computacional	Dependência de um dispositivo central para fazer o registro de clientes

Ao final, é possível notar que muitos trabalhos utilizam técnicas semelhantes, como as operações XOR e as funções *hash*. Ainda, uma parte considerável adotam as mesmas técnicas, necessitando de atenção para mitigar os diversos ataques que a rede está sujeita a sofrer. Em consequência disso, alguns trabalhos acabam sofrendo ataques como roubo de dados, falsificação de usuários e até mesmo *man-in-the-middle*. Além disso, é preciso atenção na utilização dos métodos, objetivando economia em termos de recursos, caso contrário, perderão em escalabilidade. É possível destacar ainda que a solução pro-

posta no presente trabalho não depende de apenas uma entidade para autenticar todas as entidades da rede.

3. Esquema de Autenticação para IoT

O ATTACH-IoT foi constituído por quatro etapas, cada uma realizando um conjunto de atividades com o objetivo de autenticar o dispositivo IoT ao *broker* MQTT. A primeira é o processo de inicialização (i), onde o cliente e o servidor estabelecem os parâmetros iniciais para a autenticação. A segunda é o procedimento de autenticação (ii), na qual o cliente realiza a sua devida autenticação e define suas credenciais. A terceira etapa é o processo de acordo de chave (iii), onde será definida a chave utilizada durante o algoritmo de criptografia. Por último, a quarta etapa é de utilização das credenciais (iv) para publicar uma mensagem na rede. O detalhamento sobre cada uma dessas fases será apresentado a seguir. Para identificar essas operações são utilizados alguns símbolos, como XOR (Operação OU exclusivo), $\|$ (Concatenação), H (Operação Hash) e PUF (Função PUF).

3.1. Inicialização do Esquema de Autenticação

Nesta fase o servidor e o cliente negociam as primeiras informações que serão compartilhadas entre ambos. A técnica de PUF apresenta uma ideia promissora para a autenticação de dispositivos em rede, gerando valores únicos e impossíveis de serem clonados por atacantes [Bolotnyy and Robins 2007]. A assinatura gerada pelo algoritmo PUF é feita através do par desafio-resposta, onde uma entidade envia um pacote contendo um valor como desafio, o destinatário recebe, executa o PUF e responde enviando o seu resultado. O valor enviado no início pode ser, por exemplo, um número gerado randomicamente.

Para efeitos de prova de conceito, neste trabalho é realizada uma simulação desta estratégia, onde o cliente responde ao desafio enviando um *mock* do algoritmo PUF. Além disso, o servidor gera um identificador único que será compartilhado com o cliente para garantir a identificação entre ambos. Isso é necessário para mitigar ataques direcionados para autenticação mútua. As três primeiras trocas de mensagens serão feitas utilizando um canal seguro de comunicação utilizando o protocolo Transport Layer Security (TLS), isso se dá pela necessidade de transmitir dados sigilosos entre o servidor e cliente e não permitir que sejam trafegados em texto limpo. A Figura 1 descreve a fase de inicialização.

(1) A fim de se apresentar para o servidor e iniciar a troca de mensagens, o cliente envia um pacote com o texto $\{Hello\}$ para o servidor;
(2) Com o objetivo de executar a função PUF no lado do cliente, o servidor gera o desafio Ci e o envia dentro do canal seguro criado pelo TLS;
(3) O cliente ao receber o desafio do servidor, utiliza-o para executar o algoritmo PUF e gerar o valor $Ri = PUF(Ci)$, e então envia o resultado para o servidor dentro do canal seguro criado pelo TLS;
(4) O servidor recebe a resposta do cliente ao seu desafio e então gera o seu identificador Sid e envia para o cliente (feita por um canal seguro). Ele faz isso pois, futuramente o cliente precisará se apresentar para o servidor como um usuário legítimo, para isso, ele utilizará o Sid para iniciar essa verificação. Ao final o servidor armazena os valores $\{Ci, Ri\ e\ Sid\}$;

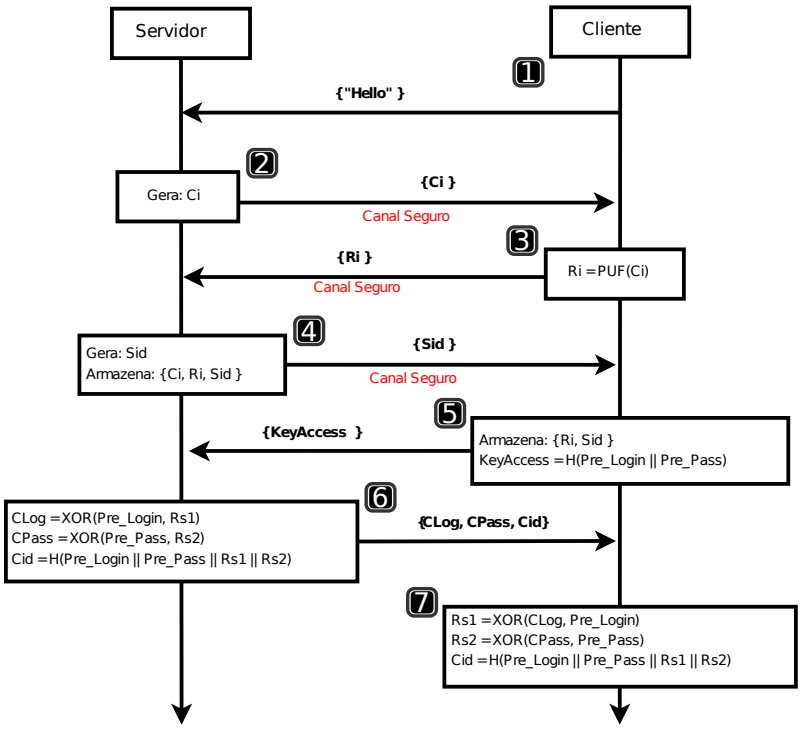

Figura 1. Fase de Inicialização

(5) O cliente recebe o Sid do servidor e o armazena, pois ele será utilizado posteriormente na quarta fase do esquema;
Na sequência, é realizada uma operação *hash* com os valores que ele recebeu do usuário, $KeyAccess = H(Pre\_Login||Pre\_Pass)$;

(6) O resultado é enviado para o servidor, este faz a verificação se o valor recebido é aceito, caso for, ele vai gerar dois valores aleatórios, $Rs1$ e $Rs2$. Estes serão utilizados para iniciar a criação dos login, senha e ID do cliente;
Após gerados os valores, são realizadas as operações $CLog = Pre\_Login \oplus Rs1$, $CPass = Pre\_Pass \oplus Rs2$ e $Cid = H(Pre\_Login||Pre\_Pass||Rs1||Rs2)$;
O cliente realiza essas operação XOR com o objetivo de esconder os valores aleatórios gerados. Para isso, ele usa o Pre_Login e o Pre_Pass na operação, que são valores conhecidos tanto pelo cliente quanto pelo servidor. Quando os resultados das operações XOR ($CLog$ e $CPass$) chegarem ao servidor, para que ele consiga obter os dados aleatórios gerados, realiza-se a operação XOR reversa com os valores que ele recebeu do cliente utilizando os dados comuns entre eles (Pre_Login e o Pre_Pass);
Feito isso o servidor vai enviar esses valores para o cliente;

(7) O cliente recebe e armazena os três valores $\{CLog, CPass, Cid\}$; Realiza as seguintes operações para extrair os valores de $Rs1$ e $Rs2$: $Rs1 = CLog \oplus Pre\_Login$ e $Rs2 = CPass \oplus Pre\_Pass$. Ou seja, o que o cliente faz é executar as operações XOR de maneira contrária ao que o servidor fez, obtendo assim os valores aleatórios que fora gerados ($Rs1$ e $Rs2$);
Feito isso ele calcula $Cid = H(Pre\_Login||Pre\_Pass||Rs1||Rs2)$. O resultado será comparado com o hash que foi recebido com o objetivo de verificar se ambos os valores são iguais e que não houve nenhuma mudança nos dados trocados entre cliente e servidor.

3.2. Autenticação do Cliente

Com os dados anteriores armazenados, o cliente inicia o processo de autenticação, onde serão geradas as devidas credenciais e, ao final, estará devidamente autenticado pelo servidor. A Figura 2 exibe o fluxo de pacotes.

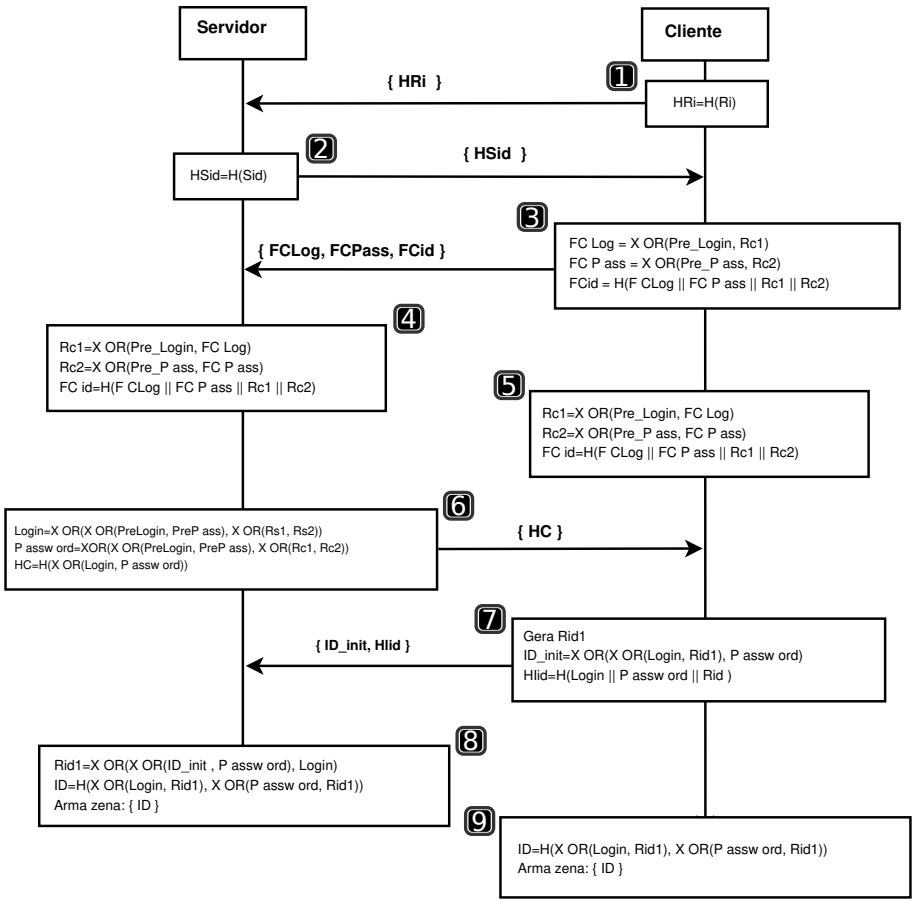

Figura 2. Fase de Autenticação

(1) A fim de se apresentar para o servidor como um cliente legítimo, o cliente gera um *hash* do seu Ri e envia para o servidor validar;

(2) O servidor assim que recebe o *hash* criado pelo cliente, usa o Ri que ele armazenou na etapa anterior, gera um *hash* com ele e compara com o recebido. Caso sejam iguais, o servidor gera um *hash* do seu Sid e envia para o cliente avaliar. Ele faz isso a fim de comprovar a sua legitimidade como servidor para o novo cliente;

(3) O cliente recebe o *hash* gerado pelo servidor, pega o Sid que ele armazenou na etapa anterior, gera um *hash* com ele e compara com o que ele recebeu, caso sejam iguais, a comunicação continua. Utilizando os valores $Rs1$ e $Rs2$ gerados na etapa anterior, o cliente vai gerar mais dois valores aleatórios, $Rc1$ e $Rc2$, que serão utilizados também na criação das credenciais do cliente, login, senha e ID;
Na sequência, executa as seguintes operações $FCLog = Pre\_Login \oplus Rc1$, $FCPass = Pre\_Pass \oplus Rc2$ e $FCid = H(FCLog||FCPass||Rc1||Rc2)$. Estes valores são enviados para que o servidor calcule e encontre os valores aleatórios gerados, o Rc1 e Rc2;

(4) O servidor recebe o pacote e faz as seguintes operações para obter os valores aleatórios gerados pelo cliente: $Rc1 = Pre\_Login \oplus FCLog$,

$Rc2 = Pre\_Pass \oplus FCPass$. Após isso, ele utiliza os valores obtidos para tentar obter o mesmo hash que ele recebeu no pacote, $FCid = H(FCLog||FCPass||Rc1||Rc2)$. Ao final ele irá comparar o hash recebido com o que foi obtido para garantir que os valores trocados no pacote anterior não sofreram alterações;

(5-6) O servidor e o cliente realizam as seguintes operações para gerar o login e a senha do cliente: $Login = (PreLogin \oplus PrePass) \oplus (Rs1 \oplus Rs2)$ e $Password = (PreLogin \oplus PrePass) \oplus (Rc1 \oplus Rc2)$;
Na sequência, ambos calculam o *hash* dos valores obtidos com o objetivo de fazer a validação deles, ou seja, verificar se não houve nenhuma alteração de dados nos passos anteriores: $HC = H(Login \oplus Password)$;
O servidor envia para o cliente o *hash* final e o cliente, ao receber, compara com o *hash* que ele gerou, caso sejam iguais, então o login e senha do cliente estão definidos.

Definidas as credenciais, login e senha, nesta próxima etapa o cliente e o servidor irão trocar mensagens a fim de gerar um próximo identificador único para o cliente. Este identificador será utilizado como uma outra forma de garantir a autenticidade do cliente no momento que ele for realizar a publicação de alguma mensagem. Para gerar o identificador e finalizar a fase de autenticação do cliente, serão executados os passos abaixo.

(7) Para gerar o identificador, é necessário possuir dados iniciais que auxiliem na sua criação. O cliente então gera um valor aleatório $Rid1$ e depois calcula $ID\_init = (Login \oplus Rid1) \oplus Password$, que serão utilizados para iniciar a criação do seu identificador;
Esta operação é realizada com o objetivo de esconder o valor aleatório obtido, primeiro é feita uma operação XOR entre o $Login$ e $Rid1$, após isso, ele realiza outra operação XOR entre o resultado obtido e o $Password$. Posteriormente o valor de $Rid1$ pode ser obtido fazendo operações XOR reversas utilizando os valores corretos de $login$ e $Password$;
Na sequência, o cliente gera o *hash* $HIid = H(Login||Password||Rid)$, para garantir a integridade dos dados sigilosos que foram utilizados para gerar o ID_init;
O cliente envia para o servidor $\{ID\_init, HIid\}$;

(8) O servidor ao receber o pacote enviado pelo cliente realizará a seguinte operação para encontrar o valor aleatório gerado pelo cliente, $Rid1 = (ID\_init \oplus Password) \oplus Login$;
O servidor realiza a operação inversa do que foi realizado no lado cliente, ele primeiro faz a operação XOR entre o ID_init com o $Password$) e depois e repete a operação XOR entre o resultado obtido e o $Login$;
Posteriormente, é gerado o ID do novo cliente, $ID = H((Login \oplus Rid1) \oplus (Password \oplus Rid1))$;

(9) As entidades envolvidas realizam a seguinte operação para gerar o ID do novo cliente, $ID = H((Login \oplus Rid1) \oplus (Password \oplus Rid1))$, após isso, ambas armazenam o ID do novo cliente.

Após a autenticação do cliente e o recebimento das devidas credenciais, a comunicação dentro da rede torna-se possível. Pensando em prover um nível de segurança adicional para os clientes, foi elaborado um esquema de acordo de chaves de criptografia, descrito a seguir.

3.3. Acordo de Chave de Criptografia

Neste ponto, o cliente já possui credenciais armazenadas, login, senha e ID, e ele utilizará esses dados para inicializar o acordo de chave com o servidor. Na Figura 3 é exibido o fluxo de pacotes. Semelhante aos valores secretos (Pre_Login e Pre_Pass) que foram armazenados previamente pelo usuário ao dispositivo, existirá um valor secreto também para gerar a chave de criptografia. Esse valor será chamado de Pre_Key, seguindo o padrão dos valores anteriores.

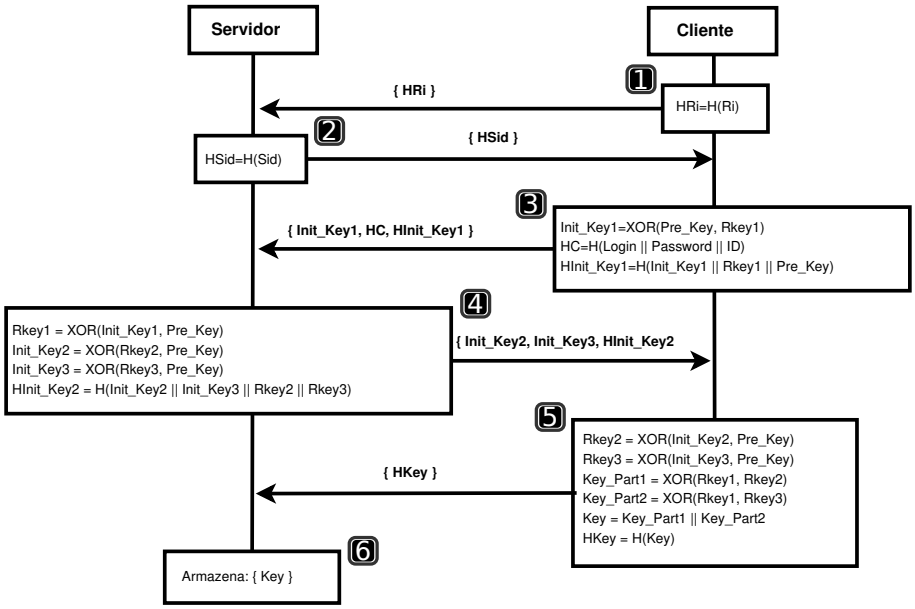

Figura 3. Fase de Acordo de Chaves

(1) A fim de se apresentar para o servidor como um cliente legítimo na rede, o cliente gera um *hash* do seu Ri e envia para o servidor validar;

(2) O servidor, assim que recebe o *hash* criado pelo cliente, usa o Ri que ele armazenou na etapa anterior, gera um *hash* com ele e compara com o recebido. Caso sejam iguais, o servidor gera um *hash* do seu Sid e envia para o cliente avaliar. Ele faz isso a fim de comprovar a sua legitimidade como servidor para o novo cliente;

(3) O cliente recebe o *hash* gerado pelo servidor, e utiliza o Sid que ele armazenou na etapa anterior, gerando um *hash* com ele e compara com o que ele recebeu, caso sejam iguais, a comunicação continua. Para gerar a chave que será utilizada na criptografia simétrica, é necessário antes gerar alguns dados que iniciarão esse processo. Para isso, o cliente gera um valor aleatório chamado $Rkey1$ e então executa $Init\_Key1 = Pre\_Key \oplus Rkey1$, $HC = H(Login \mid\mid Password \mid\mid ID)$ e $HInit\_Key1$ = $H(Init\_Key1 \mid\mid Rkey1 \mid\mid Pre\_Key)$;

O $Init\_Key1$ é usado para esconder o valor aleatório que foi criado, o HC é criado a fim do servidor validar as credenciais do cliente para confirmar que ele é legítimo e o $HInit\_Key1$ é usado para tentar garantir a integridade dos dados sigilosos usados;

Em seguida, o cliente envia uma solicitação ao servidor para iniciar o acordo da chave de criptografia, o pacote enviará os dados $\{Init\_Key1$, HC, $HInit\_Key1\}$;

(4) O servidor recebe o pacote do cliente, e para encontrar o valor aleatório gerado por ele, executa a operação: $Rkey1 = Init\_Key1 \oplus Pre\_Key$. Após isso, ele realiza o devidos hashs com os valores secretos armazenados e compara com os *hash* recebidos do cliente, o HC e $HInit\_Key1$;
Após as confirmações, o servidor então gera dois valores aleatórios $Rkey2$ e $Rkey3$. Estes serão utilizados em conjunto com o $Rkey1$ para criar a nova chave de criptografia. Depois executa: $Init\_Key2 = Rkey2 \oplus Pre\_Key$, $Init\_Key3 = Rkey3 \oplus Pre\_Key$, $HInit\_Key2 = H(Init\_Key2||Init\_Key3||Rkey2||Rkey3)$;
Feito isso o servidor envia para o cliente um pacote contendo $\{Init\_Key2$, $Init\_Key3$, $HInit\_Key2\}$;

(5) O cliente ao receber o pacote do servidor realiza as seguintes operações para encontrar os valores aleatórios gerados pelo servidor: $Rkey2 = Init\_Key2 \oplus Pre\_Key$ e $Rkey3 = Init\_Key3 \oplus Pre\_Key$. Semelhante aos passos anteriores, o cliente consegue encontrar os valores aleatórios realizando a operação reversa do XOR. Após isso, é gerado o *hash* $HInit\_Key2$ com os valores encontrados e então é comparado com o que foi recebido;
Ambas as entidades realizam as seguintes operações: $Key\_Part1 = Rkey1 \oplus Rkey2$ e $Key\_Part2 = Rkey1 \oplus Rkey3$. Elas geram esses valores com o objetivo de formar a nova chave de criptografia, Key_Part1 e Key_Part2 são as duas metades da chave, então as entidades primeiro geram esses dois valores para posteriormente concatená-los;
Após isso, é gerada a chave de criptografia executando $Key = Key\_Part1 \; || \; Key\_Part2$;
Para confirmar a nova chave de criptografia, o cliente gera um *hash* da chave $HKey = H(Key)$ e envia para o servidor $\{HKey\}$;

(6) O servidor gera um *hash* da chave também e compara com o *hash* recebido do cliente a fim de validar a informação final. Caso os *hash* sejam iguais, ambos armazenam a nova chave de criptografia criada.

Com o final do processo de autenticação, o cliente estando registrado, autenticado e com suas credenciais definidas, quando ele for fazer alguma publicação no *broker* MQTT, ele precisará receber autorização do *broker* para realizar essa publicação. Para isso, será explicada a última etapa do esquema onde o cliente solicitará autorização ao *broker* para publicar uma mensagem.

3.4. Autorização para Publicação

Após ter se autenticado, o cliente possui neste momento os seguintes dados sigilosos: login, senha, ID, Ri e Sid . Como estes dados não foram transmitidos publicamente na rede e estão protegidos, eles serão utilizados para este próximo passo. Esses dados serão as credenciais utilizadas para o cliente conseguir acesso ao servidor. A Figura 4 exibe o fluxo de pacotes.

(1) O cliente inicia a comunicação com o servidor enviando um pacote contendo $\{$ Sid, "$CONNECT$" $\}$. O cliente envia este pacote para informar ao servidor que ele precisa publicar uma mensagem na rede, para isso ele informa o Sid do servidor e envia uma mensagem $CONNECT$ indicando o início dessa etapa;

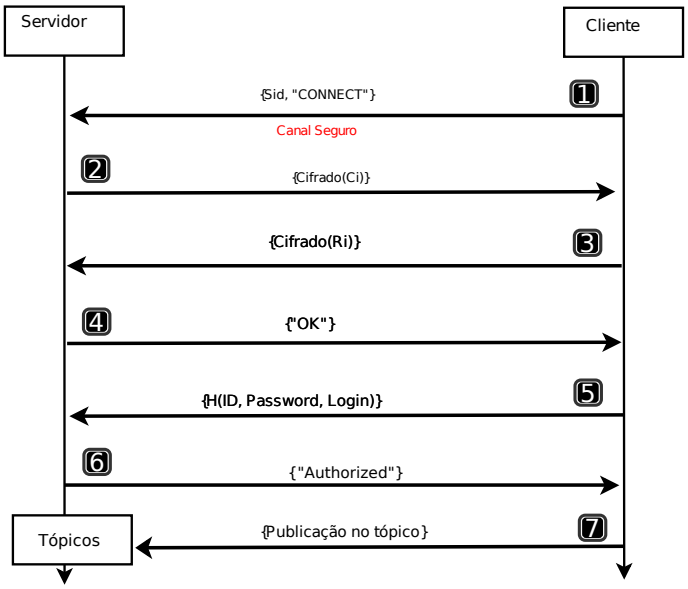

Figura 4. Autorização para Publicação

(2) O servidor ao receber o pacote e percebe que se trata de uma requisição para o cliente publicar uma mensagem, envia um pacote contendo $\{Cifrado(Ci)\}$. Ele faz isso a fim de verificar se o cliente possui o mesmo Ri armazenado no servidor, que corresponde ao identificador gerado pela função PUF na primeira etapa do esquema;
(3) O cliente recebe o desafio Ci, pega o Ri que está armazenado e responde enviando $\{Cifrado(Ri)\}$;
(4) O servidor recebe o pacote e verifica se a resposta corresponde ao valor que está armazenado localmente, caso seja, ele continua o processo enviando um pacote de confirmação $\{"OK"\}$. Ele faz isso para garantir ao cliente que a sua resposta ao desafio está correta e que agora ele pode enviar as demais credenciais que ele possui;
(5) Após receber a resposta positiva o cliente agora pega as demais credenciais armazenadas e envia o seguinte pacote $\{H(ID, Password, Login)\}$. Isto permite que o servidor realize o procedimento de validação da sua identidade;
(6) O servidor aguarda a resposta do cliente e realiza a mesma função *hash* com os dados armazenados e compara o pacote recebido. Caso sejam iguais, o servidor envia uma mensagem de confirmação para o cliente;
(7) O cliente então, após receber a confirmação do servidor, publica a mensagem no tópico em que se inscreveu.

4. Implementação do ATTACH-IoT

O esquema proposto foi implementado em um ambiente virtual utilizando contêineres. O código do esquema proposto se encontra disponível em um repositório aberto[1] e de livre acesso. Utilizou-se a ferramenta Docker para criar e gerenciar os contêineres. Criou-se um contêiner para hospedar o *broker*, outro para fazer o papel de *publisher* e mais dois para atuarem como *subscribers*.

A Figura 5 mostra as saídas obtidas ao executar o servidor, onde são exibidas as informações de todos os clientes. Os dados dos usuários são organizados em formato

[1] `https://github.com/degemarps/Esquema-Autenticacao-Mestrado`

de dicionário, onde a chave correspondente a cada informação é o Sid (identificador do servidor). Percebe-se que ao final foram autenticados três clientes, o publicador e os dois *subscribers* do protocolo MQTT. No bloco de informações no início da figura são mostradas as informações do primeiro usuário autenticado, o *publisher*, e logo em seguida, dos dois *subscribers*.

```
root@d66079c74e19:/home# ./server.py
Cis: {58677213: 14372829}
Ris: {58677213: '592063e50118040c2000112b'}
Sids: [58677213]
Logins: {58677213: 3542712644}
Passwords: {58677213: 3558906955}
Keys: {58677213: 'c5947aa150209099'}

Cis: {58677213: 14372829,
      95729497: 20867350}
Ris: {58677213: '592063e50118040c2000112b',
      95729497: '592063e50118040c2000112b'}
Sids: [58677213, 95729497]
Logins: {58677213: 3542712644,
         95729497: 3499121582}
Passwords: {58677213: 3558906955,
            95729497: 3523028878}
Keys: {58677213: 'c5947aa150209099',
       95729497: '725a804b886d10e1'}

Cis: {58677213: 14372829,
      95729497: 20867350,
      54264336: 35559421}
Ris: {58677213: '592063e50118040c2000112b',
      95729497: '592063e50118040c2000112b',
      54264336: '37c0480063021011988c0095'}
Sids: [58677213, 95729497, 54264336]
Logins: {58677213: 3542712644,
         95729497: 3499121582,
         54264336: 3496290837}
Passwords: {58677213: 3558906955,
            95729497: 3523028878,
            54264336: 3601995723}
Keys: {58677213: 'c5947aa150209099',
       95729497: '725a804b886d10e1',
       54264336: 'b22a9cdf73a53dec'}
```

Figura 5. Servidor/*Broker*

```
root@c7becdcdca36:/home# ./publisher.py
Ci:  b'35559421'
Ri:  37c0480063021011988c0095
Sid:  54264336
Login:  3496290837
Password:  3601995723
Key:  b22a9cdf73a53dec
Publicacao Realizada
```

Figura 6. *Publisher*

```
root@cc5230157056:/home# ./subscriber.py
Ci:  b'14372829'
Ri:  592063e50118040c2000112b
Sid:  58677213
Login:  3542712644
Password:  3558906955
Key:  c5947aa150209099
Connected with result code 0
Hello world!
```

Figura 7. Primeiro *Subscriber*

```
root@841eb79dc8fe:/home# ./subscriber.py
Ci:  b'20867350'
Ri:  592063e50118040c2000112b
Sid:  95729497
Login:  3499121582
Password:  3523028878
Key:  725a804b886d10e1
Connected with result code 0
Hello world!
```

Figura 8. Segundo *Subscriber*

A Figura 6 mostra as saídas obtidas ao executar o publicador. É possível notar que as informações apresentadas condizem com o que foi exibido na Figura 5, quando o publicador se autenticou por último no servidor. As saídas mostram as suas respectivas informações após o cliente ter se autenticado e logo após é mostrada uma mensagem informando que uma publicação foi realizada.

Nas Figuras 7 e 8 são exibidas as saídas dos dados que foram gerados durante o processo de autenticação realizadas pelos *subscribers* no ATTACH-IoT. As saídas apresentadas exibem os resultados dos processos de autenticação dos *subscribers*, mostrando os dados gerados, e logo após ele realiza uma inscrição em um tópico no *broker* MQTT. O resultado *connected with result code 0* informa que o cliente conseguiu se inscrever em um tópico com sucesso. Por último, é exibida uma mensagem *Hello World!*, mensagem publicada pelo *publisher* indicando que ele se autenticou e passou pelo processo de autorização para publicar.

5. Validação formal do ATTACH-IoT

A validação formal do ATTACH-IoT foi realizada no *Scyther*. Para cada uma das fases implementadas no esquema foi desenvolvido um teste formal de segurança. As Figuras 9 e 10 mostram os resultados dos testes feitos para a fase de setup e de autenticação, respectivamente. Na fase de setup os resultados exibem que foram cumpridos os requisitos definidos. Nesta fase os dispositivos precisam proteger os dados sigilosos PrePass, $PreLogin$, $Rs2$, $Rs1$, Sid e Ci. De acordo com os testes, o algoritmo conseguiu atender a essa característica. Além disso, os resultados mostraram que esta fase foi capaz de atender aos requisitos de sincronização na comunicação, acordo das mensagens, autenticação mútua entre as partes comunicantes e garantia de que as entidades estão utilizando o mesmo protocolo. Na segunda fase do esquema, os resultados dos testes também foram positivos. Todas as informações sigilosas, que precisavam permanecer protegidas durante toda a comunicação, receberam aprovação da ferramenta, garantindo assim, proteção de dados. Com isso, a fase de autenticação conseguiu atender aos objetivos de segurança estabelecidos.

Claim				Status	Comments
setup	S	setup,S2	Secret Rs2	Ok	No attacks within bounds.
		setup,S3	Secret Rs1	Ok	No attacks within bounds.
		setup,S4	Secret Sid	Ok	No attacks within bounds.
		setup,S5	Secret Ci	Ok	No attacks within bounds.
		setup,S6	Secret PrePass	Ok	No attacks within bounds.
		setup,S7	Secret PreLogin	Ok	No attacks within bounds.
		setup,S8	Alive	Ok	No attacks within bounds.
		setup,S9	Weakagree	Ok	No attacks within bounds.
		setup,S10	Niagree	Ok	No attacks within bounds.
		setup,S11	Nisynch	Ok	No attacks within bounds.
	C	setup,C2	Secret PrePass	Ok	No attacks within bounds.
		setup,C3	Secret PreLogin	Ok	No attacks within bounds.
		setup,C4	Secret Rs2	Ok	No attacks within bounds.
		setup,C5	Secret Rs1	Ok	No attacks within bounds.
		setup,C6	Secret Sid	Ok	No attacks within bounds.
		setup,C7	Secret Ci	Ok	No attacks within bounds.
		setup,C8	Alive	Ok	No attacks within bounds.
		setup,C9	Weakagree	Ok	No attacks within bounds.
		setup,C10	Niagree	Ok	No attacks within bounds.
		setup,C11	Nisynch	Ok	No attacks within bounds.

Figura 9. Fase de Setup

Claim				Status	Comments
setup	S	setup,S2	Secret Rs2	Ok	No attacks within bounds.
		setup,S3	Secret Rs1	Ok	No attacks within bounds.
		setup,S4	Secret Sid	Ok	No attacks within bounds.
		setup,S5	Secret Ri	Ok	No attacks within bounds.
		setup,S6	Secret PrePass	Ok	No attacks within bounds.
		setup,S7	Secret PreLogin	Ok	No attacks within bounds.
		setup,S8	Secret Rid	Ok	No attacks within bounds.
		setup,S9	Secret Rc2	Ok	No attacks within bounds.
		setup,S10	Secret Rc1	Ok	No attacks within bounds.
		setup,S11	Alive	Ok	No attacks within bounds.
		setup,S12	Weakagree	Ok	No attacks within bounds.
		setup,S13	Niagree	Ok	No attacks within bounds.
		setup,S14	Nisynch	Ok	No attacks within bounds.
	C	setup,C2	Secret Rid	Ok	No attacks within bounds.
		setup,C3	Secret Rc2	Ok	No attacks within bounds.
		setup,C4	Secret Rc1	Ok	No attacks within bounds.
		setup,C5	Secret Sid	Ok	No attacks within bounds.
		setup,C6	Secret Ri	Ok	No attacks within bounds.
		setup,C7	Secret PrePass	Ok	No attacks within bounds.
		setup,C8	Secret PreLogin	Ok	No attacks within bounds.
		setup,C9	Secret Rs2	Ok	No attacks within bounds.
		setup,C10	Secret Rs1	Ok	No attacks within bounds.
		setup,C11	Alive	Ok	No attacks within bounds.
		setup,C12	Weakagree	Ok	No attacks within bounds.
		setup,C13	Niagree	Ok	No attacks within bounds.
		setup,C14	Nisynch	Ok	No attacks within bounds.

Figura 10. Fase de Autenticação

As Figuras 11 e 12 exibem, respectivamente, os resultados dos testes feitos para a fase de acordo de chaves e para o processo autorização para publicação de mensagens. Na fase de acordo de chaves foram obtidos resultados positivos. De acordo com os re-

sultados, todos os dados sigilosos que necessitavam estar protegidos passaram nos testes, o que significa que esta fase garante proteção de dados. Sendo assim, o acordo de chaves do esquema ocorre de forma segura e oferecendo proteção de dados às informações sigilosas compartilhadas. Na fase de autorização, todas as credenciais necessárias para esse processo foram utilizadas de forma segura, em nenhum momento elas foram expostas durante a comunicação entre o cliente e o servidor. Logo, o processo de busca por autorização para publicação de mensagens ocorre de forma segura e oferece proteção às credenciais que estão armazenadas nos dispositivos.

Claim				Status	Comments
criacaoChave	X	criacaoChave,X2	Secret Rkey3	Ok	No attacks within bounds.
		criacaoChave,X3	Secret Rkey2	Ok	No attacks within bounds.
		criacaoChave,X4	Secret Sid	Ok	No attacks within bounds.
		criacaoChave,X5	Secret Ri	Ok	No attacks within bounds.
		criacaoChave,X6	Secret PreKey	Ok	No attacks within bounds.
		criacaoChave,X7	Secret ID	Ok	No attacks within bounds.
		criacaoChave,X8	Secret Password	Ok	No attacks within bounds.
		criacaoChave,X9	Secret Login	Ok	No attacks within bounds.
		criacaoChave,X10	Secret Rkey1	Ok	No attacks within bounds.
		criacaoChave,X11	Alive	Ok	No attacks within bounds.
		criacaoChave,X12	Weakagree	Ok	No attacks within bounds.
		criacaoChave,X13	Niagree	Ok	No attacks within bounds.
		criacaoChave,X14	Nisynch	Ok	No attacks within bounds.
	Y	criacaoChave,Y2	Secret Rkey1	Ok	No attacks within bounds.
		criacaoChave,Y3	Secret Sid	Ok	No attacks within bounds.
		criacaoChave,Y4	Secret Ri	Ok	No attacks within bounds.
		criacaoChave,Y5	Secret PreKey	Ok	No attacks within bounds.
		criacaoChave,Y6	Secret ID	Ok	No attacks within bounds.
		criacaoChave,Y7	Secret Password	Ok	No attacks within bounds.
		criacaoChave,Y8	Secret Login	Ok	No attacks within bounds.
		criacaoChave,Y9	Secret Rkey3	Ok	No attacks within bounds.
		criacaoChave,Y10	Secret Rkey2	Ok	No attacks within bounds.
		criacaoChave,Y11	Alive	Ok	No attacks within bounds.
		criacaoChave,Y12	Weakagree	Ok	No attacks within bounds.
		criacaoChave,Y13	Niagree	Ok	No attacks within bounds.
		criacaoChave,Y14	Nisynch	Ok	No attacks within bounds.

Figura 11. Fase Acordo de Chaves

Claim				Status	Comments
autorizacao	S	autorizacao,S2	Secret Login	Ok	No attacks within bounds.
		autorizacao,S3	Secret Password	Ok	No attacks within bounds.
		autorizacao,S4	Secret Id	Ok	No attacks within bounds.
		autorizacao,S5	Secret Ri	Ok	No attacks within bounds.
		autorizacao,S6	Secret Ci	Ok	No attacks within bounds.
		autorizacao,S7	Secret Sid	Ok	No attacks within bounds.
		autorizacao,S8	Alive	Ok	No attacks within bounds.
		autorizacao,S9	Weakagree	Ok	No attacks within bounds.
		autorizacao,S10	Niagree	Ok	No attacks within bounds.
		autorizacao,S11	Nisynch	Ok	No attacks within bounds.
	C	autorizacao,C2	Secret Login	Ok	No attacks within bounds.
		autorizacao,C3	Secret Password	Ok	No attacks within bounds.
		autorizacao,C4	Secret Id	Ok	No attacks within bounds.
		autorizacao,C5	Secret Ri	Ok	No attacks within bounds.
		autorizacao,C6	Secret Ci	Ok	No attacks within bounds.
		autorizacao,C7	Secret Sid	Ok	No attacks within bounds.
		autorizacao,C8	Alive	Ok	No attacks within bounds.
		autorizacao,C9	Weakagree	Ok	No attacks within bounds.
		autorizacao,C10	Niagree	Ok	No attacks within bounds.
		autorizacao,C11	Nisynch	Ok	No attacks within bounds.

Figura 12. Processo de Autorização

Ao final dos testes, os resultados de todas as fases mostraram excelente performance do esquema proposto nos requisitos de segurança. Todos os códigos utilizados nas validações estão disponíveis publicamente no GitHub[2].

6. Conclusão

Atualmente a IoT está passando por um crescimento acelerado e com isso, aumenta a necessidade de existir um bom algoritmo de segurança no processo de autenticação de dispositivos. Devido a esse problema, o presente trabalho teve como objetivo apresentar o ATTACH-IoT, um esquema de autenticação e acordo de chaves para ser utilizado em conjunto com os protocolos de comunicação para IoT. O esquema composto por três fases no processo de autenticação e uma fase de autorização para publicação de mensagens. A solução foi validada formalmente com o *Scyther* e também implementada

[2]https://github.com/degemarps/Scyther-Testes-Mestrado

em um ambiente conteinerizado. Os resultados comprovaram que todas as validações formais foram executadas com sucesso e a implementação em ambiente virtualizado conseguiu executar de forma segura. Todos os experimentos de validação formal e a implementação do esquema proposto foram disponibilizados abertamente, permitindo assim avançar a literatura da área assim como a reprodutibilidade dos experimentos. Com base nos resultados, a sugestão como trabalho futuro é de implementação da solução em dispositivos IoT reais. Com isso, será possível exercitar o protótipo em cenário real de modo a obter maior confiança e acurácia na análise e interpretação dos resultados.

Referências

Bittencourt, L. et al. (2018). The internet of things, fog and cloud continuum: Integration and challenges. *Internet of Things*, 3-4:134 – 155.

Bolotnyy, L. and Robins, G. (2007). Physically unclonable function-based security and privacy in rfid systems. In *Fifth Annual IEEE International Conference PerCom*, pages 211–220.

Cremers, C. J. F. (2008). The scyther tool: Verification, falsification, and analysis of security protocols. In Gupta, A. and Malik, S., editors, *Computer Aided Verification*, pages 414–418, Berlin, Heidelberg. Springer Berlin Heidelberg.

do Prado, P. F. et al. (2021). *Mobile Edge Computing for Content Distribution and Mobility Support in Smart Cities*, pages 473–500. Springer International Publishing, Cham.

Fernandes, R. et al. (2020). S3as: uma solução de autenticação e autorização através de aplicativos de smartphones. *Revista Eletrônica Argentina-Brasil de Tecnologias da Informação e da Comunicação*, 3(1).

Fiorenza, M. and et al. (2021). Representação e aplicação de políticas de segurança em firewalls de redes híbridas. In *Anais do XXXIX Simpósio Brasileiro de Redes de Computadores e Sistemas Distribuídos*, pages 490–503, Porto Alegre, RS, Brasil. SBC.

Garcia-Morchon, O. et al. (2019). Internet of Things (IoT) Security: State of the Art and Challenges. RFC 8576.

Hassija, V. et al. (2019). A survey on iot security: Application areas, security threats, and solution architectures. *IEEE Access*, 7:82721–82743.

il Bae, W. and Kwak, J. (2017). Smart card-based secure authentication protocol in multi-server IoT environment. *Multimedia Tools and Applications*, 79(23-24):15793–15811.

Kreutz, D. et al. (2020). Auth4app: Protocols for identification and authentication using mobile applications. In *Anais do XX SBSEG*, pages 422–435, Porto Alegre, RS, Brasil. SBC.

Lee, J. et al. (2019). Secure three-factor authentication protocol for multi-gateway IoT environments. *Sensors*, 19(10):2358.

Oh, J. et al. (2021). A secure and lightweight authentication protocol for iot-based smart homes. *Sensors*, 21(4).

Pisani, F. et al. (2020). Fog computing on constrained devices: Paving the way for the future iot. *Advances in Edge Computing: Massive Parallel Processing and Applications*, 35:22.

Rührmair, U. and Holcomb, D. E. (2014). Pufs at a glance. In *2014 Design, Automation Test in Europe Conference Exhibition (DATE)*, pages 1–6.

Zhu, F. et al. (2019). A lightweight rfid mutual authentication protocol with puf. *Sensors*, 19(13).