# CACIC-DevKit: Construção de Sistemas IoT com Políticas de Acesso Customizáveis e Segurança por Hardware

**Guilherme A. Thomaz**[1], **Matheus B. Guerra**[1],
**Matteo Sammarco**[2] **e Miguel Elias M. Campista**[1]

[1]Grupo de Teleinformática e Automação (GTA)
Universidade Federal do Rio de Janeiro (UFRJ)
[2]Pesquisador Independente

{guiaraujo,barreira,miguel}@gta.ufrj.br, matteosam@hotmail.it

***Resumo.*** *Os enclaves de memória garantem a segurança de dados em nuvem na presença de atacantes com acesso privilegiado aos servidores. A diversidade de aplicações na Internet das Coisas dificulta o desenvolvimento de sistemas com enclaves. Este trabalho propõe o CACIC-DevKit: uma ferramenta para desenvolver sistemas de Internet das Coisas utilizando a arquitetura segura CACIC, proposta em trabalho anterior. O diferencial da ferramenta é a flexibilidade na escolha das fontes de dados, das tarefas de processamento e do banco de dados adequado ao seu caso de uso. Para demonstrar a usabilidade, um sistema para redes elétricas inteligentes é desenvolvido com dispositivos reais, interfaces gráficas de usuário e bancos de dados comerciais.*

***Abstract.*** *Memory enclaves secure data in clouds in the presence of attackers with privileged access to servers. The diversity of applications in the Internet of Things (IoT) makes it difficult to develop enclave systems. This work proposes CACIC-DevKit: a tool for developing IoT systems using the CACIC architecture, proposed in our previous work. The differential of the tool is the flexibility in choosing the appropriate data sources, processing tasks, and database for your use case. To demonstrate the usability, a system for smart-grids is developed with real devices, graphical user interfaces, and commercial databases.*

## 1. Introdução

Os sensores da Internet das Coisas (*Internet of Things* – IoT) coletam dados sensíveis como sinais biomédicos, padrões de consumo e imagens de câmeras de segurança para prover mais conforto e automação. Esses dados são tipicamente processados e armazenados em servidores em nuvem devido à restrição de poder de processamento dos sensores. Como consequência, os provedores em nuvem podem obter vantagens financeiras ao utilizar os dados dos clientes de forma indevida [Zegzhda et al. 2017]. Diferente dos mecanismos criptográficos tradicionais, os ambientes de execução confiáveis (*Trusted Execution Environment* – TEE) utilizam recursos de *hardware* para garantir segurança. Em um TEE, nem mesmo *softwares* privilegiados, como sistemas operacionais, podem comprometer a confidencialidade e a integridade dos dados.

Este trabalho foi realizado com recursos do CNPq, Coordenação de Aperfeiçoamento de Pessoal de Nível Superior – Brasil (CAPES) – Código de Financiamento 001, FAPERJ e FAPESP (2015/24494-8, 2018/23292-0, 2015/24485-9, 2014/50937-1).

A arquitetura CACIC (Controle de Acesso Confiável Usando Enclaves a Dados em Nuvem da Internet das Coisas) foi proposta e avaliada em termos de segurança e desempenho em trabalhos anteriores [Thomaz et al. 2022b, Araujo et al. 2019]. Nessa arquitetura, o servidor usa o Intel *Software Guard Extensions* (SGX) para isolar a execução de código em enclaves de memória [Costan and Devadas 2016]. Através de um procedimento de atestação, uma medida criptográfica do enclave é assinada e enviada para o cliente, que verifica a autenticidade do código. Ademais, os resultados de um processamento em enclave são encriptados com uma chave de selagem antes de serem escritos em disco. O Intel SGX oferece instruções especiais para inicialização do enclave, atestação e selagem, de modo que nenhum outro *software* comprometa as prerrogativas de segurança. O objetivo do CACIC é garantir que o servidor cumpra as políticas de acesso aos dados IoT na nuvem.

Este trabalho propõe, implementa e documenta o CACIC-DevKit: uma ferramenta para que desenvolvedores utilizem a arquitetura CACIC para construir sistemas IoT. Um diferencial da ferramenta é a flexibilidade para escolher as fontes de dados, as interfaces com o usuário, as tarefas de processamento e os bancos de dados mais adequados ao caso de uso. A principal contribuição do trabalho é a abstração dos mecanismos de segurança e o controle de acesso em enclave do CACIC para desenvolvedores não especializados em computação confiável. Para demonstrar o CACIC-DevKit, este trabalho desenvolve um sistema para que usuários disponibilizem padrões de consumo de energia sem revelar dados detalhados que comprometam sua privacidade. Os resultados confirmam que a ferramenta é compatível com sensores, interfaces gráficas e bancos de dados com ampla adoção comercial.

Este trabalho está estruturado da seguinte forma. A Seção 2 introduz o CACIC-DevKit, sua arquitetura e funcionalidades. Já a Seção 3 demonstra a aplicação desenvolvida com o CACIC-DevKit, enquanto a Seção 4 discute os trabalhos relacionados. Por fim, a Seção 5 conclui o trabalho e aponta direções futuras de pesquisa.

## 2. Ferramenta CACIC-DevKit

Esta seção resume a arquitetura do CACIC e descreve as funcionalidades da ferramenta CACIC-DevKit para que o desenvolvedor crie seus próprios sistemas IoT seguros [Thomaz et al. 2022b, Thomaz et al. 2022a]. O repositório com os códigos do CACIC-DevKit está disponível em `https://github.com/GTA-UFRJ/CACIC-DevKIt`. Já a documentação está disponível em `https://cacic-devkit.readthedocs.io/en/latest/index.html`. A licença de *software* escolhida é a do MIT, permitindo modificação e redistribuição de cópias desde que contenham as informações de direito de uso, disponíveis no repositório. Por fim, o vídeo demonstrando a ferramenta baseada em um caso de uso está disponível em `https://youtu.be/CFEsD-25Mp0`.

### 2.1. Arquitetura

A Figura 1 ilustra os componentes do CACIC e as mensagens de publicação (`M[pub]`), consulta (`M[con]`) e registro (`M[reg]`) enviadas ao servidor pelos clientes. Cada cliente possui um número hexadecimal de oito dígitos (`ID`) e uma chave secreta de comunicação (`CC`), compartilhada com o enclave do servidor na etapa de registro.

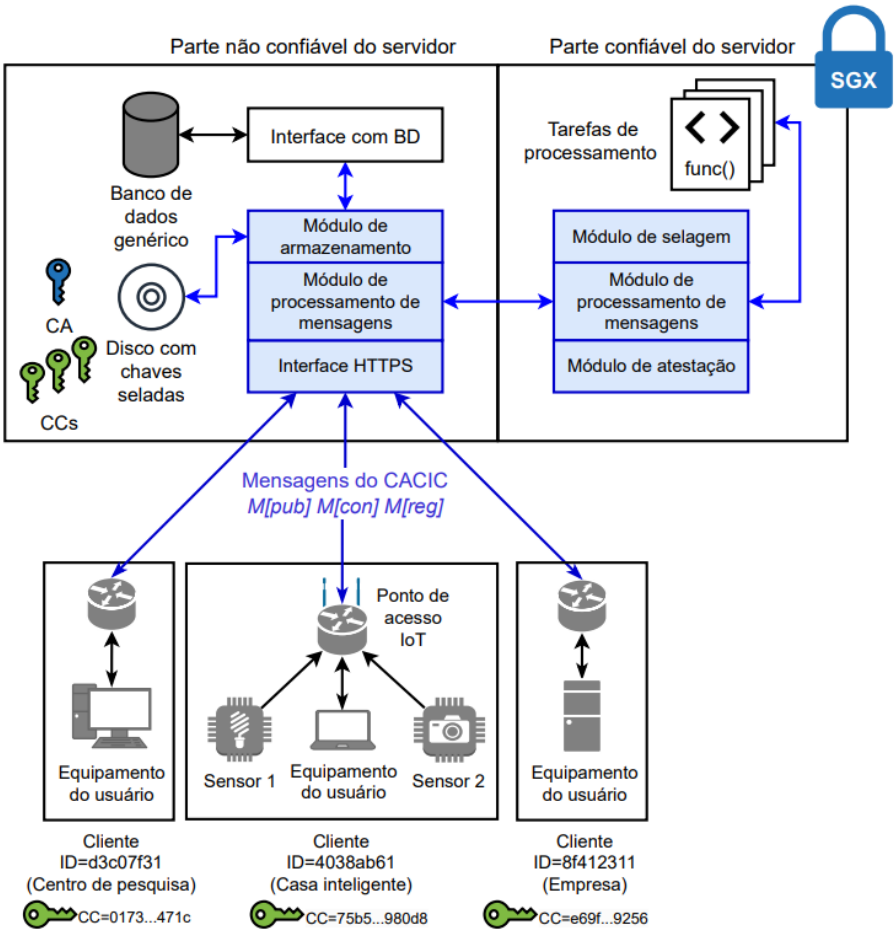



**Figura 1. Os componentes e interfaces em azul fazem parte do núcleo do CACIC, ou seja, não dependem do caso de uso. A arquitetura é flexível quanto às tarefas de processamento, ao banco de dados e às fontes de dados.**

Dependendo do caso de uso, os clientes podem ser casas inteligentes que publicam e gerenciam dados de sensores ou instituições interessadas em obter dados. As mensagens de publicação contêm uma requisição de publicação e uma lista de `IDs` de clientes com permissão de acesso ao dado, ambas encriptadas com a chave `CC` [Karjoth et al. 2002]. O campo `tipo` da mensagem de publicação indica se o servidor publicará um novo dado ou executará uma tarefa de processamento antes de publicar o resultado. O resultado e a lista de permissões de acesso são armazenados encriptados com uma chave de armazenamento do servidor (`CA`). Algumas tarefas de processamento podem exigir que o servidor utilize dados publicados previamente, como na construção de modelos preditivos, por exemplo. Já as mensagens de consulta contêm uma requisição de consulta, utilizada para localizar o dado no banco de dados. Entretanto, antes de retornar o dado requisitado, o enclave do servidor o decripta e verifica se a lista de permissões de acesso incluem o cliente. O mesmo mecanismo de controle de acesso é utilizado para decidir se um dado publicado anteriormente pode ser utilizado em uma determinada tarefa de processamento.

O servidor decripta dados e permissões de acesso apenas dentro do enclave, isolando o controle de acesso e o processamento dos dados até mesmo de atacantes com acesso privilegiado ao servidor. As chaves usadas para encriptar as mensagens dos clien-

**Tabela 1. As funções em preto devem ser programadas de acordo com o caso de uso, enquanto que as funções em azul fazem parte do núcleo do CACIC-DevKit.**

| Número | Função | Parte da arquitetura que chama a função | Funcionalidade |
|---|---|---|---|
| 1 | *publish_db()* | Interface com BD | Publica um dado no BD com um comando |
| 2 | *query_db()* | Interface com BD | Consulta um dado do BD com um comando |
| 3 | *multi_query_db()* | Interface com BD | Consulta múltiplos dados do BD com um comando |
| 4 | *nome_da_tarefa()* (usuário define) | Módulo de processamento das mensagens | Processa dados de acordo com a requisição recebida na M[pub] e gera resultado |
| 5 | *enclave_query_db()* | Tarefa de processamento | Consulta um dado publicado anteriormente para uso na tarefa de processamento |
| 6 | *enclave_multi_query_db()* | Tarefa de processamento | Análogo à função acima, mas para múltiplos dados |
| 7 | *enclave_get_payload()* | Tarefa de processamento<br>Módulo de proc. das mens. | Os dados são acompanhados de metadados. A função retorna o dado bruto. |
| 8 | *client_publish()* | Equipamento do usuário<br>Sensor/ponto de acesso | Monta uma M[pub] e envia para o servidor |
| 9 | *client_query()* | Equipamento do usuário<br>Sensor/ponto de acesso | Monta uma M[con] e envia para o servidor |
| 10 | *client_register()* | Equipamento do usuário | Monta uma M[reg] e envia para o servidor |

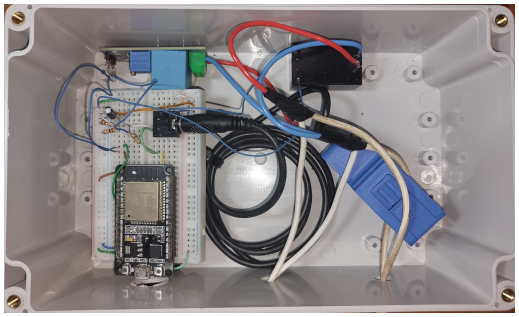
tes e os dados armazenados (`CC` e `CA`, respectivamente) são seladas no disco, podendo ser recuperadas apenas pelo enclave. Na etapa de registro, o servidor atesta sua autenticidade para o cliente ao enviar para uma medição criptográfica assinada, realizada por instruções do SGX. A arquitetura CACIC não impõe restrições computacionais ao sensor, pois a montagem e encriptação da mensagem pode ser delegada para um ponto de acesso. Os componentes em azul na Figura 1 são pré-definidos pelo CACIC, mas a arquitetura pode ser implementada para diferentes fontes de dados, interfaces com o usuário, bancos de dados e tarefas de processamento, motivando o desenvolvimento do CACIC-DevKit.

## 2.2. Funcionalidades

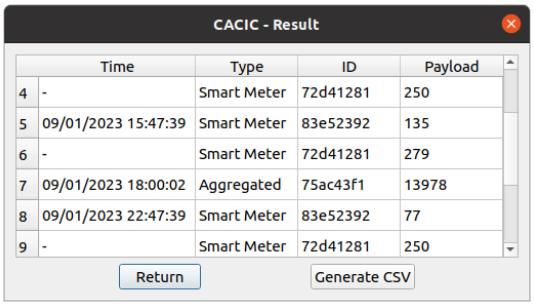
As funcionalidades do CACIC-DevKit são oferecidas aos desenvolvedores para construção de sistemas IoT através das funções enumeradas na Tabela 1. Enquanto as funções em azul compõem o núcleo do CACIC, as funções em preto devem ser programadas pelo desenvolvedor a depender do caso de uso. Os passos resumidos para desenvolver um sistema IoT com as funcionalidades propostas são os seguintes:

1. programar as tarefas de processamento (Função 4) e associá-las a um *tipo* de mensagem de publicação. O desenvolvedor pode chamar funções auxiliares (Funções 5, 6 e 7) do núcleo do CACIC-DevKit para consultar dados prévios, por exemplo. Essas funções já realizam as verificações de segurança e controle de acesso;
2. programar as interfaces com o banco de dados (Funções 1, 2 e 3);
3. programar a interface do cliente, que pode chamar as Funções 8, 9 e 10;
4. programar a fonte de dados, que pode chamar as Funções 8, 9 e 10. Um sensor pode enviar diretamente as mensagens de publicação ao servidor ou um ponto de acesso pode interceptar dados de sensores e construir as mensagens.

Uma vantagem do desacoplamento entre o núcleo e o caso de uso é a abstração dos mecanismos de segurança, simplificando o desenvolvimento de sistemas seguros por usuários não especializados na área. Por exemplo, quando o núcleo recebe uma mensagem de publicação, o enclave decripta a requisição de publicação com a chave `CC`, e identifica a tarefa de processamento programada pelo desenvolvedor (Função 4), utilizando o campo *tipo*. O resultado do processamento é encriptado com a `CA` e inserido no banco



(a)




(b)

**Figura 2. (a) Fotografia do sensor. (b) Janela da aplicação na qual o usuário pode filtrar atributos e consultar dados do servidor relacionados ao consumo de energia de um cliente no último mês, por exemplo. Os campos vazios indicam que o cliente pode não enviar informações de horário.**

de dados com a Função 1, programada pelo desenvolvedor. Nesse exemplo, o desenvolvedor se concentra nos códigos da tarefa de processamento e da interface com o banco de dados, enquanto que o núcleo da ferramenta lida com a segurança de forma transparente. Todos os códigos executados dentro do enclave dispõem de funções oferecidas pelo kit de desenvolvimento de *software* (*Software Development Kit* – SDK) do SGX. A documentação do CACIC-DevKit oferece informações detalhadas sobre as funções, como os argumentos, retornos e arquivos onde as funções devem ser implementadas. Ademais, o repositório conta com programas prontos de interfaces gráficas e de linha de comando, banco de dados SQLite e tarefas de agregação de dados numéricos, que exemplificam as funcionalidades do CACIC-DevKit com um caso de uso.

## 3. Caso de uso e demonstração

Esta seção descreve um caso de uso da ferramenta baseado em redes elétricas inteligentes (*smart grids*). O objetivo é demonstrar a compatibilidade da ferramenta com interfaces de usuário, sensores com capacidade de processamento restrita e bancos de dados amplamente adotados em sistemas comerciais. O caso de uso auxilia os usuários interessados em aprender a ferramenta, oferecendo códigos que exemplificam o uso das funções. Trabalhos anteriores confirmam a importância de garantir a confidencialidade de dados de consumo de energia em nuvem, pois podem revelar informações privadas como o número de pessoas usando um dispositivo em um dado momento [Eibl and Engel 2014]. Nesse cenário, o CACIC permite que companhias de energia ou centros de pesquisa obtenham acesso aos padrões de consumo de energia sem acessar os dados individuais gerados por medidores inteligentes (*smart meters*). Para isso, o cliente pode emitir uma mensagem de publicação requisitando ao servidor que agregue amostras no enclave e disponibilize o resultado do processamento para as instituições escolhidas.

A demonstração do sistema usa um sensor, visto na Figura 2(a), projetado com uma placa de desenvolvimento ESP32 contendo um microcontrolador Dual-Core de 32 bits, 240MHz, 420KB de RAM e conectividade WiFi 802.11b/g/n. O sensor também usa um medidor de corrente SCT-013 e um medidor de tensão ZMPT101B. O *firmware* do sensor, programado em C++, processa os sinais de tensão e corrente com auxílio da biblioteca EmonLib e envia periodicamente o consumo usando o protocolo HTTP para um ponto de acesso. Apesar da comunicação entre sensor e ponto de acesso utilizar

HTTP, a arquitetura define que a comunicação entre o ponto de acesso e o servidor deve utilizar HTTPS, conforme Figura 1. A arquitetura é agnóstica ao protocolo utilizado para comunicação em rede local, pois o modelo de atacante não considera ataques no cliente. Um Raspberry Pi Model-B com uma CPU Quad-Core de 64 bits, 3,2GHz e 1GB de memória RAM executa um *software* de ponto de acesso WLAN, em C++, para interceptar dados do sensor e construir as mensagens de publicação do CACIC. O servidor é uma máquina com 20 CPUs Intel i9-10900, 2,8GHz, suporte ao SGX e 32GB de RAM, que também hospeda um banco de dados SQLite. O cliente utiliza um computador Linux rodando uma aplicação de gerenciamento com uma interface de linha de comando e uma interface gráfica de usuário, implementada com o *Framework* Qt 6.4.1. A aplicação de gerenciamento do cliente conta com as funcionalidades básicas de registro, publicação e consulta, bem como funcionalidades específicas do caso de uso, como configuração remota do ponto de acesso. Vale ressaltar que a divisão entre as responsabilidades do sensor, do ponto de acesso e da aplicação do cliente é flexível ao caso de uso.

A demonstração começa com o registro do cliente usando a aplicação de gerenciamento. Através da interface, o cliente escolhe um `ID` e uma chave de comunicação (`CC`), e compartilha essas informações com o enclave do servidor e com o ponto de acesso. O `ID` e a `CC` podem ser gerados executando o *hash* de um nome de usuário e de uma senha, respectivamente. A segurança do procedimento de geração de chaves não faz parte do escopo do artigo. Em seguida, o cliente utiliza a aplicação para selecionar as permissões de acesso para cada tipo de mensagem de publicação a ser enviada. A implementação do caso de uso define dois tipos de requisições de publicação de dados: `publicação_consumo_medido` e `publicação_consumo_agregado`. A aplicação do cliente e o ponto de acesso armazenam um dicionário, associando cada tipo de mensagem de publicação com uma lista de permissões de acesso. O formato de armazenamento das permissões de acesso, do `ID` e da `CC` podem variar de acordo com a implementação do caso de uso.

A partir de então, o ponto de acesso pode receber amostras do sensor e utilizar a função `client_publish()` (Seção 2.2) para encaminhar para o servidor uma mensagem do tipo `publicação_consumo_medido`. Não há nenhuma tarefa de processamento associada a essa mensagem, de modo que o servidor apenas insere o dado encriptado no banco de dados. Em seguida, a aplicação do cliente pode enviar uma `publicação_consumo_agregado` para o servidor calcular o consumo total de energia e disponibilizar o resultado para uma instituição, por exemplo. Nesse caso, uma tarefa de processamento no enclave do servidor usa a função `enclave_multi_query_db()` (Seção 2.2) para obter dados prévios do tipo `publicação_consumo_medido` e somá-los. Se outro cliente enviar uma `publicação_consumo_agregado`, o servidor se recusa em executar agregações com dados cujo o acesso não foi concedido pelo proprietário do dado. A aplicação também oferece uma interface para enviar uma mensagem de consulta de dados, filtrando parâmetros como os tipos e os `ID`s. A Figura 2(b) apresenta uma tela da aplicação do usuário com o resultado de uma consulta válida, na qual apenas os dados com acesso permitido ao usuário são retornados pelo enclave do servidor. A aplicação permite exportar os dados no formato de valores separados por vírgula (*Comma Separated Values* – CSV) para processamento em outros *softwares*. O vídeo de demonstração apresenta os comandos de instalação, outras telas da aplicação e o ambiente de teste com os equipamentos descritos.

## 4. Trabalhos relacionados

Outros trabalhos na literatura propõem sistemas para controle de acesso a dados da Internet das Coisas. Camilo *et al.* utilizam correntes de blocos (*blockchain*) e contratos inteligentes para controlar o acesso a dados de IoT em um *marketplace* [Camilo et al. 2020]. O trabalho distribui o controle de acesso entre os nós e não utiliza ambientes de execução confiáveis. Já Xiao *et al.* combinam *blockchain* com ambientes de execução confiáveis para controle de acesso [Xiao et al. 2020]. Um diferencial do trabalho é a execução de contratos de compra fora da corrente, em um enclave do SGX. Ambos os trabalhos se concentram na comercialização dos dados entre os usuários da rede.

O trabalho de Anciaux *et al.* descreve uma arquitetura de gerenciamento de dados pessoais com processamentos genéricos em nuvem [Anciaux et al. 2019]. Os autores propõem o uso do ambiente de execução confiável ARM TrustZone em dispositivos móveis e do SGX no servidor, mas não implementam a arquitetura [Ngabonziza et al. 2016, Costan and Devadas 2016]. Outro trabalho, de Carpentier *et al.*, implementa essa arquitetura baseado em uma aplicação para recompensar funcionários que utilizam bicicletas [Carpentier et al. 2022]. O sistema é formado por um núcleo, que implementa uma política de controle de acesso com SGX, e pode ser expandido com outros enclaves para processamentos mais específicos. Os autores não se concentram em um cenário IoT e assumem o uso de *smartphones*, com grande poder computacional. Trabalhos de avaliação de desempenho devem ser realizados para confirmar que os requisitos de alta vazão e baixa latência da Internet das Coisas são atendidos pela arquitetura.

A arquitetura CACIC foi proposta e avaliada em termos de vazão, latência, uso de recursos e segurança em trabalhos anteriores [Thomaz et al. 2022b, Thomaz et al. 2022a]. Este trabalho propõe, desenvolve e avalia uma ferramenta para que desenvolvedores construam sistemas IoT a partir do CACIC. Diferente de outros trabalhos, este trabalho contribui para o estado da arte fornecendo uma ferramenta documentada, com instruções para construção de sistemas IoT protegidos contra atacantes privilegiados. O CACIC-DevKit se diferencia por não depender dos componentes da arquitetura e da familiaridade do desenvolvedor com a área de segurança. O caso de uso confirma que a compatibilidade com dispositivos restritos em termos computacionais, com bancos de dados e interfaces comerciais e com tarefas de processamento flexíveis.

## 5. Conclusão

Este trabalho propõe, implementa e documenta o CACIC-DevKit, uma ferramenta para desenvolvimento de sistemas IoT genéricos e com controle de acesso por enclaves. A implementação do caso de uso de gerenciamento de dados de consumo de energia confirma que as funcionalidades são compatíveis com sensores, interfaces e bancos de dados comerciais. A demonstração no Salão de Ferramentas do SBRC utilizará o ponto de acesso, o sensor e a ferramenta de gerenciamento de dados referentes ao caso de uso descrito na Seção 3. O servidor com SGX, instalado no Grupo de Teleinformática e Automação, irá receber as requisições de publicação e consulta de dados remotamente, sendo necessária conexão com a Internet.

Como trabalhos futuros, é prevista a extensão da arquitetura para proteção contra ataques no lado do cliente. Também pretende-se estender a ferramenta para construir

sistemas que utilizem processamento distribuído entre enclaves e para cenários nos quais o cliente deseja publicar suas próprias tarefas de processamento.

## Referências


Anciaux, N., Bonnet, P., Bouganim, L., Nguyen, B., Pucheral, P., Popa, I. S., and Scerri, G. (2019). Personal data management systems: The security and functionality standpoint. *Information Systems*, 80:13–35.

Araujo, V., Mitra, K., Saguna, S., and Åhlund, C. (2019). Performance evaluation of fiware: A cloud-based iot platform for smart cities. In *Journal of Parallel and Distributed Computing*, volume 132, pages 250–261. Elsevier.

Camilo, G. F. et al. (2020). AutAvailChain: Disponibilização segura, controlada e automática de dados IoT usando corrente de blocos. *SBRC*.

Carpentier, R., Thiant, F., Popa, I. S., Anciaux, N., and Bouganim, L. (2022). An extensive and secure personal data management system using sgx. In *25th International Conference on Extending Database Technology*.

Costan, V. and Devadas, S. (2016). Intel sgx explained. Cryptology ePrint Archive, Paper 2016/086. `https://eprint.iacr.org/2016/086`.

Eibl, G. and Engel, D. (2014). Influence of data granularity on nonintrusive appliance load monitoring. In *Proceedings of the 2nd ACM workshop on Information hiding and multimedia security*, pages 147–151.

Karjoth, G., Schunter, M., and Waidner, M. (2002). Privacy-enabled services for enterprises. In *Proceedings. 13th International Workshop on Database and Expert Systems Applications*, pages 483–487. IEEE.

Ngabonziza, B., Martin, D., Bailey, A., Cho, H., and Martin, S. (2016). Trustzone explained: Architectural features and use cases. In *2016 IEEE 2nd International Conference on Collaboration and Internet Computing (CIC)*, pages 445–451. IEEE.

Thomaz, G. A., Guerra, M. B., Detyniecki, M., Sammarco, M., and Campista, M. E. M. (2022a). Tamper-proof access control for IoT clouds using enclaves. Technical report, Universidade Federal do Rio de Janeiro (UFRJ).

Thomaz, G. A., Guerra, M. B., Sammarco, M., and Campista, M. E. M. (2022b). Cacic: Controle de acesso confiável usando enclaves a dados em nuvem da internet das coisas. In *Anais do XL Simpósio Brasileiro de Redes de Computadores e Sistemas Distribuídos*, pages 573–586. SBC.

Xiao, Y., Zhang, N., Li, J., Lou, W., and Hou, Y. T. (2020). Privacyguard: Enforcing private data usage control with blockchain and attested off-chain contract execution. In *Computer Security–ESORICS 2020: 25th European Symposium on Research in Computer Security, ESORICS 2020, Guildford, UK, September 14–18, 2020, Proceedings, Part II 25*, pages 610–629. Springer.

Zegzhda, D. P., Usov, E., Nikol'skii, A., and Pavlenko, E. Y. (2017). Use of intel sgx to ensure the confidentiality of data of cloud users.