# CAUSP LOCK

Henrique S. Souza, Mateus Kosicov, Matheus Guidoni

16 de agosto de 2025

## 1 Introdução

Este relatório documenta o Sistema de Controle de Acesso CAUSP-LOCK, desde sua concepção de alto nível até sua implementação prática. Trata-se de um conjunto de sistemas de hardware e software suficientes para garantir o acesso de múltiplos usuários a um mesmo espaço físico, nesse caso, a sala de apoio à amamentação e regulação sensorial da Faculdade de Direito da USP, localizada em seu prédio histórico, no centro de São Paulo e destinada especialmente à comunidade autista e neurodivergente. O nome "CAUSP"de CAUSP-LOCK é referência ao Coletivo Autista da USP, do qual o Henrique S. Souza faz parte.

O hardware consiste primordialmente em uma fechadura eletrônica capaz de ler e interpretar QR Codes em um protocolo próprio. Ela é composta por três principais componentes: (i) um microcontrolador ESP32-CAM, (ii) uma câmera OV2640, e (iii) uma fechadura elétrica. O ESP32-CAM controla a câmera, que, por sua vez, efetua a captura das imagens dos QR Codes, e, caso o acesso seja autorizado, aciona a abertura da fechadura elétrica. Há um circuito elétrico de interface entre o ESP32-CAM e a fechadura elétrica, necessário para o acionamento do mecanismo, pois o microcontrolador é um dispositivo de baixa potência, com saída de 3.3V, enquanto que a fechadura demanda cerca de 12V e 1A por 1s.

O software é constituído por um (i) frontend comum, disponibilizado na forma de uma página web ou um aplicativo mobile, (ii) uma API de criação de QR Codes e (iii) um banco de dados relacional. O cliente deve cadastrar-se pelo frontend; após o cadastro ser validado, segundo uma política administrativa, e registrado no banco de dados ele terá o poder de solicitar ao sistema permissão para acessar o espaço físico. Se concedida, o que pode ou não ser automático a depender do administrador, a API gera um QR Code de acesso e o disponibiliza pelo frontend.

Para promover segurança do espaço físico, o CAUSP-LOCK gera QR Codes de acesso de uso único, temporário e assinados digitalmente, com algoritmo criptograficamente seguro, o que garante autenticidade, integridade e irretratabilidade dos acessos. A chave criptográfica empregada na assinatura é compartilhada entre o microcontrolador e o backend da aplicação assincronamente. Portanto, a tranca eletrônica funciona offline, sem necessidade de internet para autenticar os códigos de acesso dos QR Codes.

## 2 Situação problema

Desde 2024, a Faculdade de Direito do Largo de São Francisco (SanFran) possui uma sala de apoio à amamentação e regulação sensorial (vide link da matéria), que faz parte de uma

política de inclusão e pertencimento para autistas e mães. Como contam os diretores Celso Campilongo e Ana Elisa Bechara:

> "Nossa sala é equipada com uma variedade de estímulos sensoriais controlados. Estes elementos foram projetados para ajudar à pessoa neurodivergente a se autorregular, promovendo o bem-estar emocional e cognitivo durante a jornada acadêmica e laboral"

Portanto, a sala promove o bem-estar da comunidade neurodivergente, especialmente a autista, além de gestantes. O projeto foi viabilizado graças à parceria estabelecida entre a faculdade e a Associação Nacional para Inclusão de Pessoas Autistas.

No entanto, a sala enfrenta problemas de subutilização e inacessibilidade. Por conta de a faculdade sofrer em grande medida problemas com furtos em todas as suas dependências e, por estar num prédio público, intencionalmente sem controle de acesso, ela é altamente vulnerável a mau uso e furtos de seus itens internos. Por isso, ela permanece trancada e, para acessá-la, o aluno deve pedir as chaves aos funcionários, nos corredores do 3º andar, em que a sala está localizada. Nesse contexto, há relatos de constrangimentos e dificuldades dos alunos autistas ao tentarem esse processo, por conta de alguns funcionários não concederem acesso às chaves sem um critério bem definido ou formalizado.

# 3 Proposta

Diante desse problema, esse trabalho propõe um sistema de controle de acesso baseado em uma tranca eletrônica de uso compartilhado e público, capaz de garantir a entrada facilitada, inclusiva e conveniente aos autistas e às mães no espaço sensorial, tal qual explicitado pelas figuras 1 e 2.

A tranca apresenta um mecanismo de autenticação e autorização baseado na apresentação de QR Codes de acesso à câmera embutida no dispositivo. Os usuários terão acesso a esses QR Codes por meio de um website ou aplicativo, administrado pela SanFran de forma simplificada, com o pré-cadastro de alunos neurodivergentes e mães, a fim de facilitar o acesso. Cada QR Code é assinado com um algoritmo de criptografia simétrica, com um conjunto de chaves secretas compartilhadas apenas pelo dispositivo e pelo sistema gerador dos QR Codes. A tranca eletrônica também se responsabiliza por garantir a integridade e a autenticidade dos códigos, além de impor as limitações de uso único e expiração programada. Almeja-se, com esse sistema, facilitar o acesso à sala de amamentação e regulação sensorial, minimizando o esforço e os eventuais constrangimentos aos usuários e, ao mesmo tempo, garantir a segurança e o bom uso da sala.

# 4 Arquitetura do Sistema

O sistema abrange uma variedade de dispositivos físicos, mecânicos, elétricos e eletrônicos, e softwares, conforme ilustra a figura 2. Os usuários interagem com uma interface web feita em React, por meio da qual é possível, para os clientes, cadastrar-se, solicitar acesso à sala e obter QR Codes de acesso e, para os administradores, consultar os usuários cadastrados, todo o histórico de acessos da sala, conceder ou negar acesso aos clientes, controlar a concorrência,

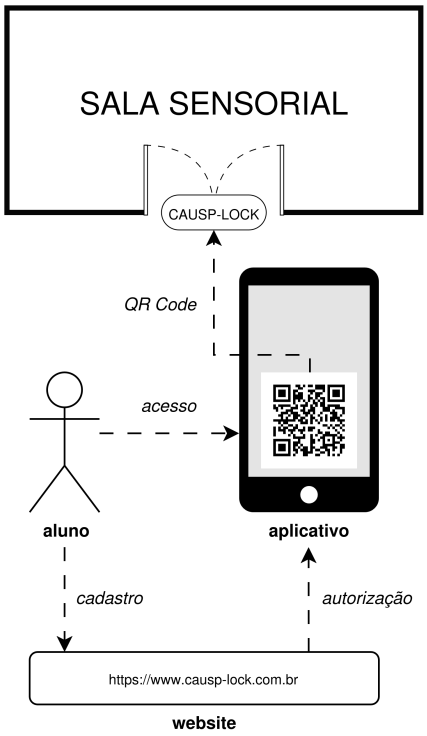


Figura 1: Visão Geral do CAUSP-LOCK

não permitindo que muitas pessoas tentem usar a sala ao mesmo tempo. O site comunica-se com uma API do python, implementada em FastAPI, para gerar os QR Codes e interagir com o banco de dados em SQL, em que se armazenam as informações dos usuários, das chaves secretas, e do histórico de uso da sala.

Ao obter os QR Codes de acesso, os clientes irão até a sala de regulação sensorial e os apresentarão à câmera OV2640, conectada ao ESP32-CAM. Então, o algoritmo fará a decodificação do QR Code em bytes, a interpretação desses bytes em dados de alto nível, a validação das assinaturas digitais HMAC-SHA1 e converterá isso em ações, por exemplo, abertura da fechadura elétrica, registro de entrada ou saída do cliente no armazenamento interno, etc.

Um aspecto importante da solução é que não há comunicação direta entre os dispositivos físicos e os serviços web. Essa escolha justifica-se por simplificar o funcionamento do acesso à sala, que, dessa forma, poderá ocorrer sem que o ESP32-CAM tenha acesso à internet. No entanto, isso também gera o trade-off de demandar dos administradores configurar os sistema web e o físico com as mesmas chaves de autenticação HMAC-SHA1.

# 5 Protocolo de Comunicação via QR Codes

A geração de QR Codes envolve codificar dados em um payload, que será então convertido numa imagem. Há diversas formas de se fazer isso, mas é necessário garantir uma codificação consistente, para que a leitura do QR Code permita decodificar o payload em seus dados originais. Para tal fim, desenvolveu-se um protocolo de comunicação via QR Codes, que define uma codificação padrão para os payloads, o que permite uma decodificação simples.

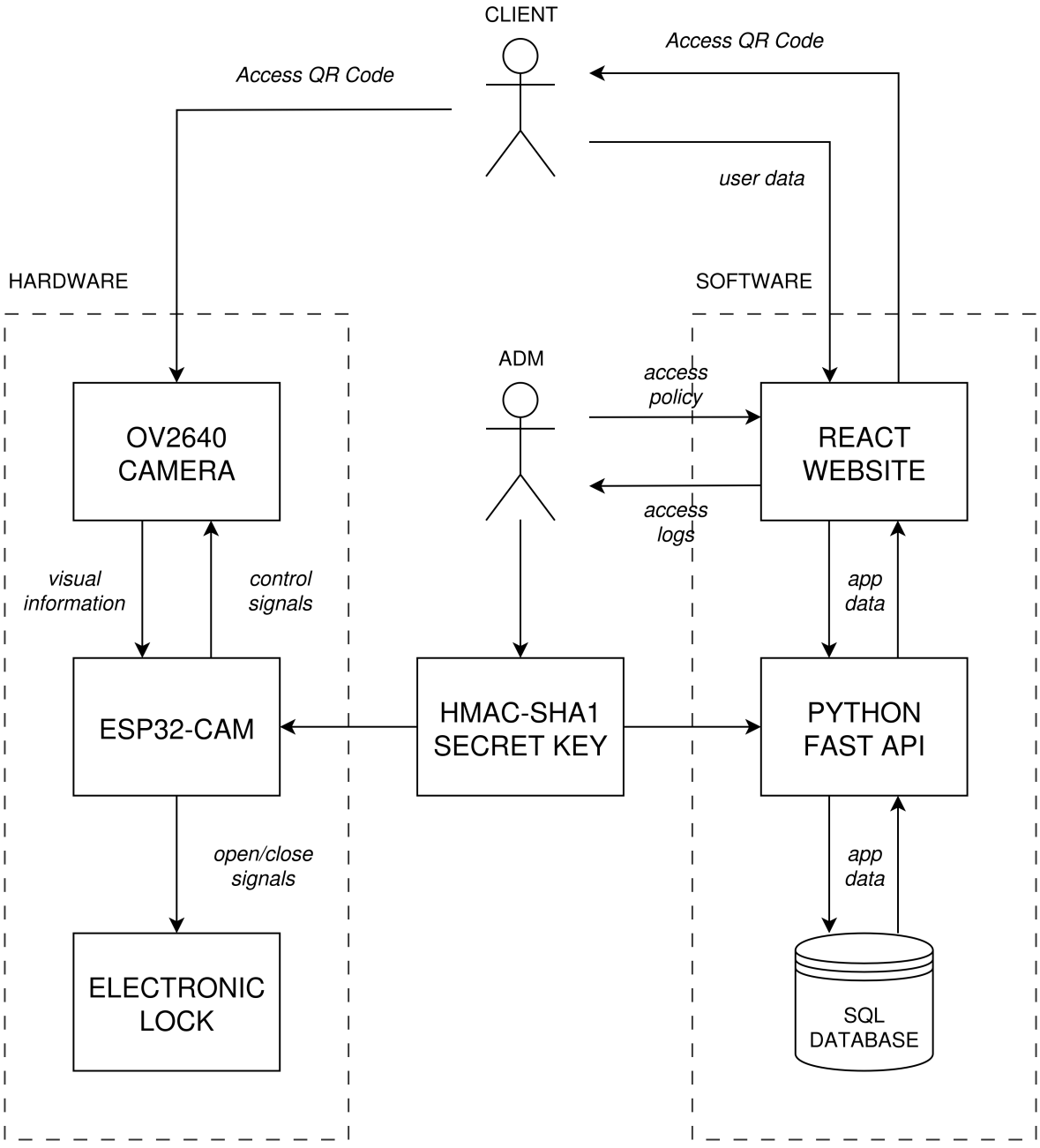


Figura 2: Arquitetura completa do sistema

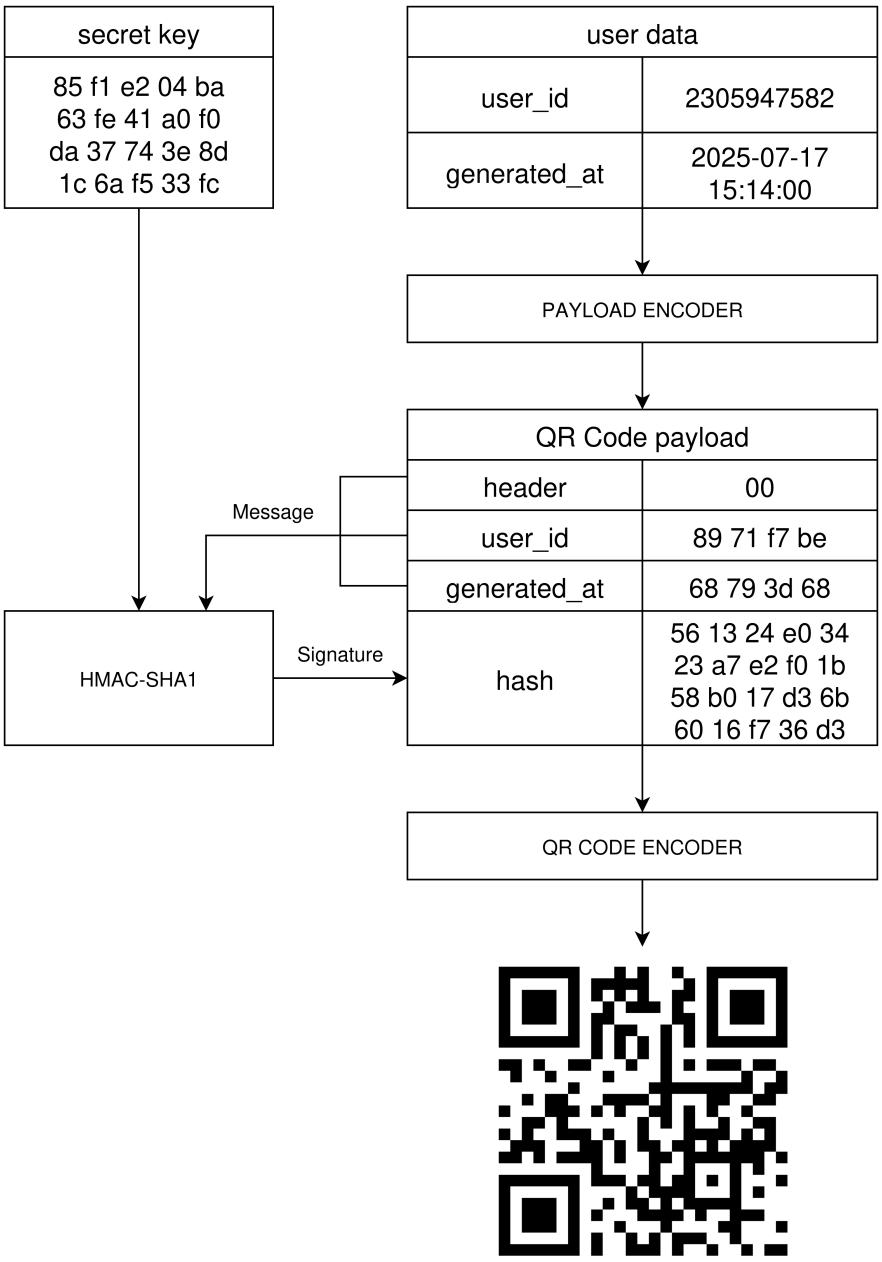


Figura 3: Codificação dos QR Codes

Conforme apresentado na figura 4, a codificação do payload, inspirada pelo datagrama do TCP, apresenta três sessões: (i) HEADER, (ii) BODY e (iii) HASH. O HEADER é o cabeçalho da mensagem e identifica seu tipo (acesso, sincronização, configuração ou depuração); seu tamanho é fixo em 1 byte. O BODY é o campo em que se colocam os dados; por exemplo, o id do usuário e o tempo em que o QR Code foi gerado, no caso de uma mensagem de acesso; seu tamanho é variado, podendo ocupar de 4 a 20 bytes. O HASH é a assinatura digital HMAC-SHA1 da mensagem e possui tamanho fixo de 20 bytes. O QR Code como um todo apresenta de 5 a 41 bytes, o total.

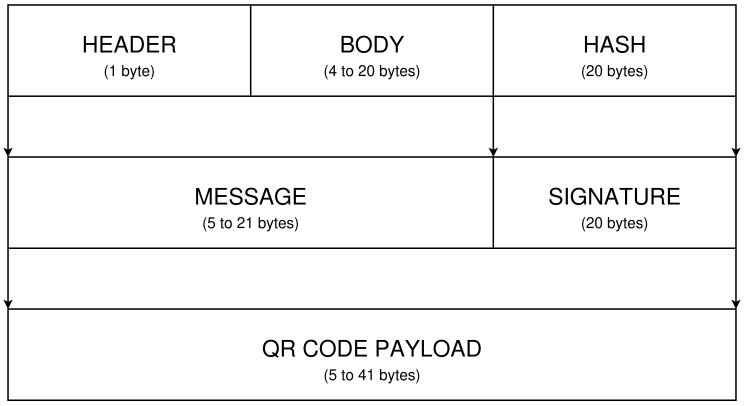


Figura 4: Codificação do payload do QR Code

Nesse contexto, denomina-se MESSAGE a concatenação de HEADER com BODY, nessa ordem (MESSAGE = HEADER + BODY), e SIGNATURE o HASH do payload. Dessa forma, a assinatura digital do QR Code considera como mensagem tanto o HEADER quanto o BODY (tipo de mensagem e dados da mensagem), o que permite verificar a integridade e autenticidade do QR Code, inviabilizando tentativas de adulteração ou falsificação de QR Codes existentes.

## 5.1 MESSAGE TYPES

Os QR Codes gerados pelo sistema web não são apenas os de acesso; há também QR Codes para sincronização de relógio do ESP32-CAM, configuração de novas chaves secretas e, por fim, QR Codes de debug, para testar o sistema sem alterar seu estado. Nosso protocolo estabelece 4 tipos de mensagem (MESSAGE_TYPE):

1. ACCESS (0): mensagens de acesso, que permitem ao cliente e administradores entrarem e saírem da sala.

2. SYNC (1): mensagens de sincronização, para configurar o relógio interno do ESP32-CAM.

3. CONFIG (2): mensagens de configuração, para configurar variáveis de ambiente e as chaves secretas HMAC-SHA1, para autenticação de mensagens.

4. DEBUG (3): mensagens de depuração, para testar o dispositivo físico, sem alterar seu estado ou seus dados armazenados.

Esses quatro tipos de mensagem recebem, cada um, um código MESSAGE_TYPE padrão. ACCESS, SYNC, CONFIG e DEBUG recebem os códigos 0, 1, 2 e 3, respectivamente.

## 5.2 OPERATION_TYPES

Enquanto o MESSAGE_TYPE determina o tipo de mensagem, o OPERATION_TYPE especifica mais a mensagem, ao explicitar que tipo de operação se deseja realizar. Por exemplo, os QR Codes de acesso são todos do tipo MESSAGE_TYPE = 0 (ACCESS), mas há um QR Code para CHECK_IN, que serve para abrir a fechadura e registar sua chegada, um CHECK_OUT, para registrar saída da sala, e um BI_ACCESS, que serve para entrar ou sair indistintamente.

| MESSAGE_TYPE | OPERATION_TYPE | ACTION_TYPE |
|---|---|---|
| ACCESS | 0 | CHECK_IN |
| ACCESS | 1 | CHECK_OUT |
| ACCESS | 2 | BI_ACCESS |
| SYNC | 0 | SET_TIME |
| CONFIG | 0 | SET_MASTER_KEY |
| CONFIG | 1 | SET_CONFIG_KEY |
| CONFIG | 2 | SET_ACCESS_KEY |
| CONFIG | 3 | SET_SYNC_KEY |
| DEBUG | 0 | BLINK_N_TIMES |
| DEBUG | 1 | BLINK_IF_SYNC |

Tabela 1: Codificação do OPERATION_TYPE

Em nosso protocolo, CHECK_IN, CHECK_OUT e BI_ACCESS são denominadas ações. O tipo de uma ação (ACTION_TYPE) é determinado pelo conjunto MESSAGE_TYPE E OPERATION_TYPE, conforme ilustram a tabela 5.2 e a figura 5.

Essa configuração permite adicionar facilmente novas ações conforme a necessidade, sem prejudicar a lógica e a numeração das já existentes. por exemplo, se desejarmos adicionar um quarto tipo de acesso, basta criar uma ação com MESSAGE_TYPE = ACCESS e OPERATION_TYPE = 3, o que não modifica a codificação das demais ações já cadastradas. Além disso, essa divisão facilita o entendimento do sistema, ao agrupar semanticamente as ações por grupo e depois especificá-las melhor com o campo OPERATION_TYPE.

## 5.3 HEADER

A seção HEADER é o cabeçalho da mensagem, que identifica seu tipo e que operação se deseja fazer. Ela contém dois campos: (i) MESSAGE_TYPE e (ii) OPERATION_TYPE. Cada campo é codificado com 4 bits, de forma que o HEADER tenha exatamente 1 byte (8 bits) de tamanho. O MESSAGE_TYPE ocupa os 4 bits mais significativos do byte de HEADER, enquanto que OPERATION_TYPE ocupa os 4 bits menos significativos. Por exemplo, a ação SET_CONFIG_KEY possui MESSAGE_TYPE = 2 = 0010 (binário) e OPERATION_TYPE = 1 = 0001 (binário), portanto, o HEADER correspondente é 0010 0001 (binário) = 21 (hexadecimal), conforme ilustra a tabela 2

## 5.4 BODY

O BODY é uma seção, de tamanho variável, que contém os dados de cada mensagem. Que campos são encontrados nela e sua interpretação dependem do MESSAGE_TYPE. Por exemplo, em QR Codes de acesso, o BODY deve conter o id do usuário (user_id) e o instante de tempo em que se gerou o código (generated_at); em QR Codes de sincronização, o BODY deve ter apenas o novo tempo, que se deseja configurar no ESP32-CAM. Os dados do BODY para cada tipo de QR Code estão descritos na tabela 3.

Na notação da tabela, X[a:b] indica os bytes índice a, a + 1, a + 2, ..., b - 3, b - 2, b - 1 de X. Portanto, BODY[0:3] são os quatro primeiros bytes do BODY; BODY[4:7], os 4 bytes

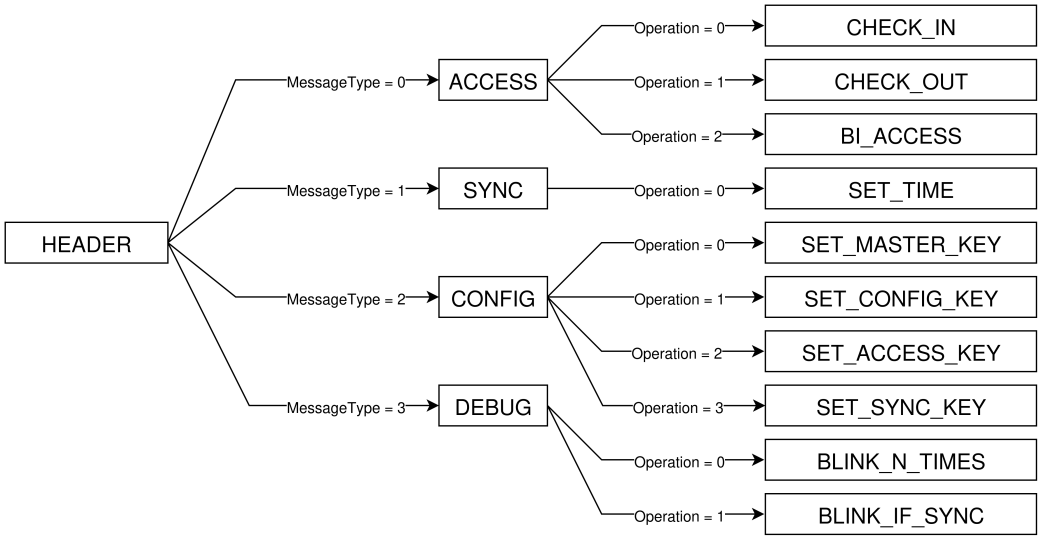


Figura 5: Codificação do HEADER

| MESSAGE_TYPE | | | OPERATION_TYPE | | HEADER | |
|---|---|---|---|---|---|---|
| name | hex | bin | hex | bin | hex | bin |
| ACCESS | 0 | 0000 | 0 | 0000 | 00 | 0000 0000 |
| ACCESS | 0 | 0000 | 1 | 0001 | 01 | 0000 0001 |
| ACCESS | 0 | 0000 | 2 | 0010 | 02 | 0000 0010 |
| SYNC | 1 | 0001 | 0 | 0000 | 10 | 0001 0000 |
| CONFIG | 2 | 0010 | 0 | 0000 | 20 | 0010 0000 |
| CONFIG | 2 | 0010 | 1 | 0001 | 21 | 0010 0001 |
| CONFIG | 2 | 0010 | 2 | 0010 | 22 | 0010 0010 |
| CONFIG | 2 | 0010 | 3 | 0011 | 23 | 0010 0011 |
| DEBUG | 3 | 0011 | 0 | 0000 | 30 | 0011 0000 |
| DEBUG | 3 | 0011 | 1 | 0001 | 31 | 0011 0001 |

Tabela 2: Codificação do HEADER

| BODY [0:3] | BODY [4:7] | BODY [8:11] | BODY [12:15] | BODY [16:19] | MESSAGE_TYPE |
|---|---|---|---|---|---|
| user_id | generated_at | - | - | - | ACCESS |
| sync_time | - | - | - | - | SYNC |
| new_key [0:3] | new_key [4:7] | new_key [8:11] | new_key [12:15] | new_key [16:19] | CONFIG |
| blink_num | - | - | - | - | DEBUG |
| sync_time | - | - | - | - | DEBUG |

Tabela 3: Codificação do BODY por MESSAGE_TYPE

seguintes, e assim por diante. Em nossa implementação, consideramos as chaves com exatos 20 bytes; mas, a princípio, esse número é arbitrário, já que o HMAC-SHA1 aceita chaves de tamanho arbitrário.Deve-se notar que há 5 campos distintos:

1. user_id (4 bytes): o identificador único do usuário, como um inteiro de 32 bits.

2. generated_at (4 bytes): o instante de tempo em que o código foi gerado, como tempo POSIX em 32 bits.

3. new_key: a nova chave HMAC-SHA1 que sobrescreverá a antiga, registrada no ESP32-CAM.

4. blink_num: número de vezes em que o ESP32-CAM deve piscar, caso se apresente esse QR Code.

5. sync_time: o instante de tempo para o qual será setado o relógio interno do ESP32-CAM no SYNC ou para teste de sincronismo no DEBUG, como tempo POSIX em 32 bits.

Um aspecto importante é que o tempo é codificado em um POSIX de apenas 4 bytes (32 bits). Isso limita a solução, pois ela só pode representar instantes de tempo aproximadamente até janeiro 2038. No entanto, para o projeto, optou-se por manter esse tempo dessa maneira mesmo assim, pois isso permitiu manter os códigos de acesso no QR Code versão 2, que é consideravelmente menor que o da versão 3; isso é essencial pois QR Codes mais simples são mais fáceis de serem lidos pela câmera OV2640.

### 5.5 HASH

O HASH é a seção do payload que contém a assinatura digital HMAC-SHA1, com exatos 20 bytes. Ela está presente em todos os MESSAGE_TYPE, exceto para os de DEBUG, por não exigirem autenticação. Ela é colocada sempre ao final do payload.

## 6 Sistema Físico

O diagrama elétrico está descrito na figura 6. O sistema fisicamente possui um ESP32-CAM, alimentado por uma fonte de 5V e 1A (máx.), ligada na rede elétrica, e conectado a uma câmera OV2640. Um pino GPIO do ESP32-CAM denominado aqui de UNLOCK é responsável por destravar a fechadura elétrica; quando em HIGH, a interpretação do sinal é a de destravar e, quando em LOW, o contrário.

No entanto, não é possível acionar a fechadura diretamente com o sinal do microcontrolador, porque, enquanto a fechadura demanda 12V e 1A por 1s, ele fornece apenas 3.3V e menos de 1A. Para contornar esse problema, emprega-se um relé eletromecânico, que interrompe condicionalmente um fio, conectado à fechadura elétrica e a uma fonte chaveada de 12V e 5A.

Mas, mesmo assim, o sinal do ESP32-CAM não é suficiente sequer para acionar o relé; por isso, empregou-se o transistor 2N2222 num circuito amplificador de sinal, utilizando a própria fonte chaveada de 12V como VCC; o sinal de 3.3V então é amplificado para cerca de 11V, que já é capaz de acionar o relé.

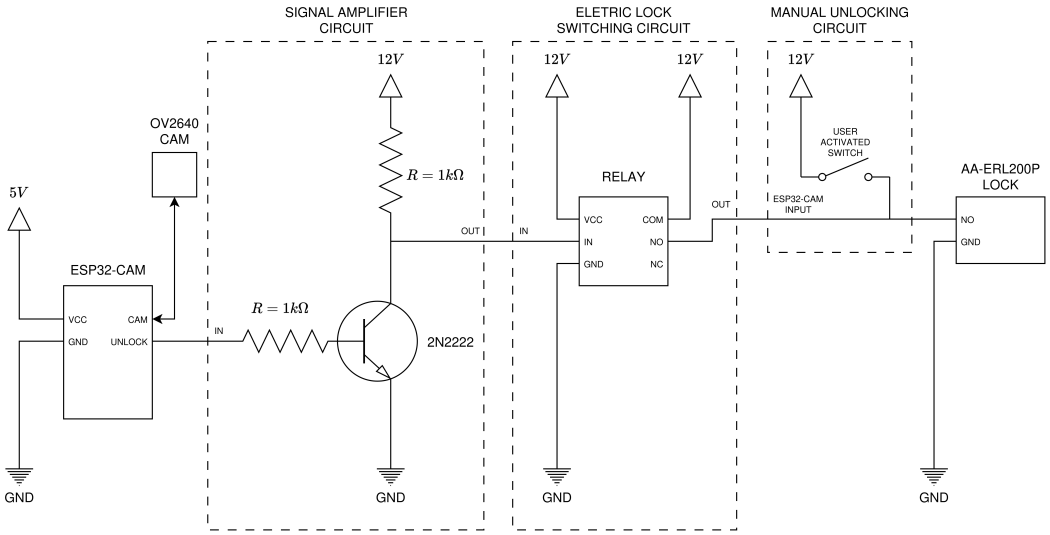


Figura 6: Diagrama elétrico

Note na figura 6 um sistema de switch, na proximidade da fechadura elétrica. Esse switch possui a função de permitir saída da sala independentemente do ESP32-CAM, por exemplo, em caso de falha dele ou erro lógico. Isso impede que o usuário fique preso na sala, após adentrá-la, sendo, portanto, um mecanismo de segurança.

# 7 Protótipo

Implementou-se o sistema CAUSP-LOCK projetado num protótipo simples, como prova de conceito do projeto. A montagem física está explicitada na figura 7, que implementa o diagrama elétrico da figura 6, apenas com a diferença de não ter o switch de acionamento interno, por simplicidade.

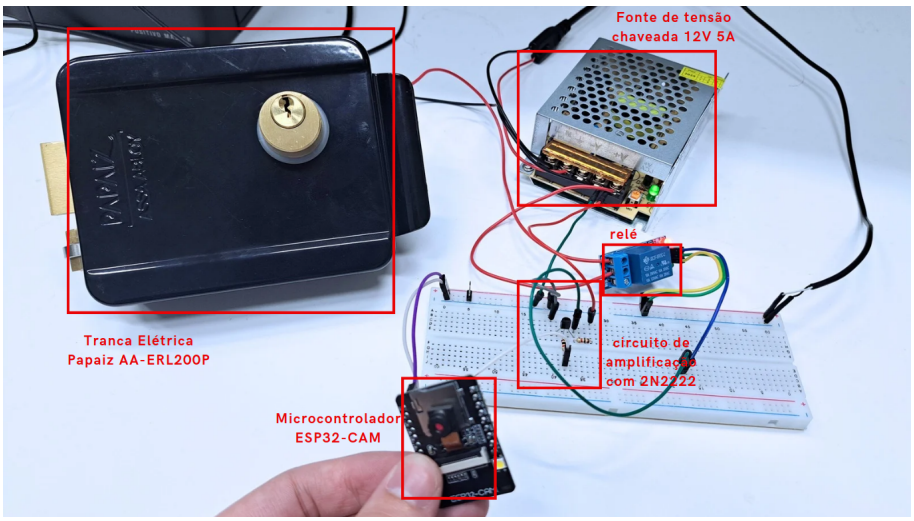


Figura 7: Protótipo como prova de conceito

Testou-se o protótipo para um caso simples, de validar um QR Code de acesso, sem se preocupar com sua expiração mas necessariamente decodificando todas suas informações, e, finalmente, acionando a fechadura elétrica, abrindo-a para o usuário. O teste foi um sucesso e pode ser visto no vídeo abaixo:

https://youtu.be/gl5iByZ4_28

# 8 ESP32-CAM

Esta sessão destina-se a explicar as especificidades técnicas do ESP32-CAM, que o tornam ideal para o projeto. O módulo ESP32-CAM é baseado no chip **ESP32-S**, que utiliza o microcontrolador **ESP32** da Espressif Systems. O componente central é o microprocessador **Tensilica Xtensa LX6**, de 32 bits, configurável para **dual-core** (inicialmente em até dois núcleos) ou **single-core** na variante ESP32-S0WD

## 8.1 Arquitetura, desempenho e recursos do processador

O ESP32-CAM utiliza o processador Xtensa LX6, que geralmente opera com dois núcleos, permitindo concorrência eficiente entre tarefas paralelas como captura de imagem, compressão JPEG e transmissão Wi-Fi. A frequência de operação é ajustável entre 80 MHz e 240 MHz, sendo este último o valor máximo recomendado. Nessa configuração, o processador alcança aproximadamente 994 pontos no benchmark CoreMark em dual-core, o que corresponde a cerca de $\approx$ 4,14 CoreMark/MHz. Sua performance teórica atinge até 600 DMIPS, e há ainda um coprocessador ULP (Ultra Low Power) integrado, capaz de realizar operações como leitura de ADCs ou comparações durante o modo de deep-sleep, sem a necessidade de ativar os núcleos principais.

O microcontrolador conta com 520 KiB de SRAM para execução de código e dados em tempo real, além de aproximadamente 448 KiB de ROM utilizada para armazenar o firmware de boot e rotinas do sistema. Há também cerca de 16 KiB de memória RTC SRAM acessível durante os modos de sono profundo, permitindo que o coprocessador ULP opere mesmo com o sistema principal desligado.

O ESP32 oferece suporte nativo ao sistema operacional FreeRTOS, permitindo execução preemptiva de tarefas com gerenciamento de threads em ambos os núcleos. Esse recurso possibilita a divisão eficiente de tarefas, como manter a comunicação via Wi-Fi em um núcleo enquanto o outro processa sinais da câmera. Adicionalmente, o chip suporta DVFS (Dynamic Voltage and Frequency Scaling), ajustando dinamicamente a frequência e a tensão de operação de acordo com a carga do sistema, o que contribui para o balanceamento entre desempenho e economia de energia.

Em termos de segurança e aceleração de tarefas criptográficas, o processador conta com boot seguro e criptografia de memória flash. Também estão integrados aceleradores de hardware para algoritmos como AES, SHA-2, RSA e curvas elípticas (ECC), além de um gerador de números aleatórios (RNG), o que garante robustez em aplicações seguras conectadas à internet.

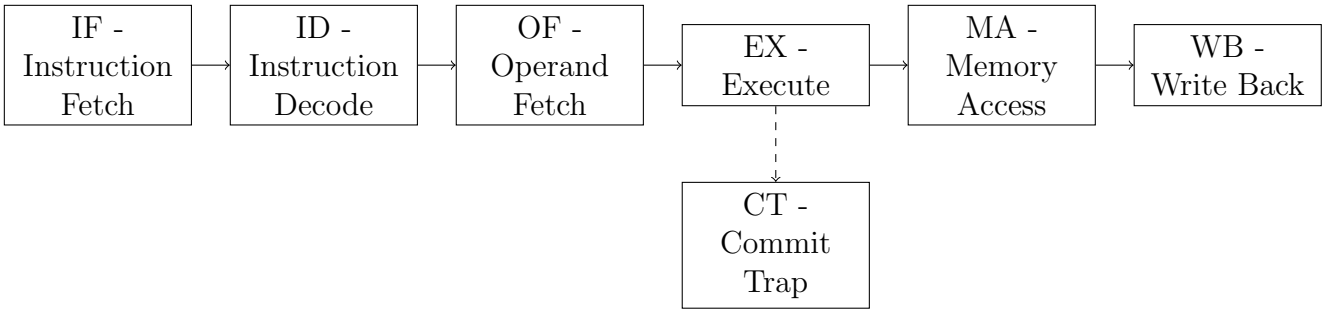


Figura 8: Pipeline simplificado de 7 estágios do Xtensa LX6.

| ***Item*** | ***Especificação*** |
|---|---|
| **CPU** | Tensilica Xtensa LX6, 32 bit, arquitetura Harvard |
| **Núcleos** | Dual-core (até 240 MHz) ou single-core (ESP32-S0WD) |
| **Desempenho** | 994 CoreMark dual-core; $\approx$ 600 DMIPS |
| **Memória interna** | 520 KiB SRAM; 448 KiB ROM; 16 KiB RTC SRAM |
| **Coprocessador ULP** | Suporte a ADC e comparações durante deep-sleep |
| **Criptografia** | AES, SHA-2, RSA, ECC; RNG embutido |
| **Modo de clock** | 80–240 MHz com DVFS |
| **Sistema operacional** | FreeRTOS com multitarefa |

Tabela 4: Especificações principais do processador Xtensa LX6 (ESP32)

## 8.2 Pipeline de 7 Estágios do Xtensa LX6

O processador **Xtensa LX6**, utilizado no ESP32, possui um pipeline de 7 estágios que permite alta frequência de operação com instruções de baixa latência. A seguir, apresenta-se um diagrama simplificado dos estágios:

O pipeline é composto pelos estágios: **IF** (Instruction Fetch), que busca a próxima instrução da memória; **ID** (Instruction Decode), que decodifica a instrução; **OF** (Operand Fetch), que busca operandos dos registradores; **EX** (Execute), que realiza a operação aritmética ou lógica; **MA** (Memory Access), que acessa a memória caso necessário; **WB** (Write Back), que escreve o resultado de volta no registrador; e **CT** (Commit Trap), que verifica e lida com interrupções e exceções.

## 8.3 Exemplo em Assembly Xtensa (manipulação de pilha)

Listing 1: Função simples em Xtensa Assembly com manipulação de pilha

```
entry    a1, 32          ; aloca 32 bytes na pilha, atualiza SP
movi     a2, 10          ; move valor 10 para a2
```

```
s32i     a2, a1, 0      ; salva a2 no topo da pilha (offset 0)
call0    func           ; chamada de funcao sem salvar janelas

func:
l32i     a3, a1, 0      ; carrega valor da pilha para a3
add      a3, a3, a3     ; duplica o valor
ret
```

**Explicação**

A execução do código se baseia em três instruções principais. A instrução `entry` é responsável por reservar o espaço necessário para variáveis locais na pilha. Em seguida, `call0` é utilizada para realizar chamadas de função de forma otimizada, sem salvar a janela de registradores. Por fim, as instruções `s32i` e `l32i` manipulam os dados diretamente, salvando e carregando valores da pilha.

## 8.4 Comparativo com ARM Cortex-M e RISC-V RV32IMC

| **Característica** | **Xtensa LX6 (ESP32)** | **ARM Cortex-M4** | **RISC-V RV32IMC** |
|---|---|---|---|
| ISA | Xtensa (proprietária) | ARMv7E-M | RV32IMC (open-source) |
| Arquitetura | Harvard, dual-core | Harvard, single-core | Von Neumann/Harvard opcional |
| Pipeline | 7 estágios | 3 a 6 estágios | 5 estágios (tip.) |
| Clock máximo | 240 MHz | 120 MHz | 100–150 MHz |
| FPU | Opcional | Presente (M4F) | Opcional |
| Consumo | Moderado (com modos sleep) | Baixo | Muito baixo |
| Debug | JTAG, OpenOCD | SWD, DWT, ITM | JTAG, OpenOCD |
| Suporte RTOS | FreeRTOS nativo | CMSIS + FreeRTOS | FreeRTOS, Zephyr |
| Licenciamento | Licença Espressif/Tensilica | ARM (licenciado) | Open-source |

Tabela 5: Comparativo entre as arquiteturas Xtensa, ARM e RISC-V

O Xtensa LX6 se destaca por seu alto desempenho e flexibilidade, sendo adequado para aplicações embarcadas com requisitos de conectividade, processamento de imagem e resposta em tempo real, como é o caso do ESP32-CAM. Em termos didáticos, permite explorar multitarefa em dual-core, modos de economia de energia, co-processadores, extensões ISA e manipulação de pilha sem instruções dedicadas.