

دانشگاه شهيد بهشتي

دانشکده مهندسی و علوم کامپیوتر

رشته مهندسی کامپیوتر

**طراحی و پیاده‌سازی اپلیکیشن اندرویدی جهت ارسال و دریافت متن رمزنگاری شده در ارتباط با سامانه سخت‌افزاری رمزنگار**

نگارش

قاسم یوسفی

استاد راهنما

دکتر راضیه سالاری‌فرد

اسفند 1400



**سپاسگزاری**

از استاد راهنمای گرانقدرم خانم دکتر راضیه سالاری‌فرد که با راهنمایی‌های دلسوزانه و حمایت‌های علمی مستمر خود مسیر انجام این پژوهش را برایم هموار ساختند، صمیمانه قدردانی نمایم؛ همچنین از خانواده عزیزم که با محبت بی‌دریغ و ایجاد آرامش و انگیزه در تمام مراحل تحصیل همراه و مشوق من بوده‌اند، نهایت سپاس و امتنان را دارم.

**چکیده**

در عصر حاضر، امنیت ارتباطات و حفظ حریم خصوصی داده‌ها به یکی از دغدغه‌های اصلی کاربران در فضای دیجیتال تبدیل شده است. با گسترش بدافزارها، کی‌لاگرها (Keyloggers) و نرم‌افزارهای جاسوسی در سیستم‌عامل‌های موبایل، اعتماد به امنیت نقاط پایانی (End-points) دشوار شده است. حتی اگر کانال ارتباطی امن باشد، در صورتی که سیستم‌عامل موبایل آلوده باشد، متن تایپ شده توسط کاربر پیش از رمزنگاری سرقت می‌شود. هدف این پروژه، طراحی و پیاده‌سازی یک سامانه سخت‌افزاری و نرم‌افزاری امن جهت ارسال و دریافت پیام‌های متنی رمزنگاری شده است که وابستگی به کیبورد مجازی تلفن همراه را از بین می‌برد.

در این پژوهش، مسئله عدم اعتماد به محیط ورودی تلفن‌های هوشمند با انتقال فرآیند تایپ و رمزنگاری به یک سخت‌افزار ایزوله حل شده است. روش پیشنهادی شامل دو بخش اصلی است: بخش نرم‌افزار که یک اپلیکیشن توسعه‌یافته با فریم‌ورک فلاتر (Flutter)است و وظیفه برقراری ارتباط سریال (USB Serial) با سخت‌افزار و انتقال پیام‌های رمز شده به بستر اینترنت را انجام می‌دهد. بخش سخت‌افزار مبتنی بر میکروکنترلر ESP32 که وظیفه دریافت ورودی از کیبورد فیزیکی (USB)، نمایش متن روی نمایشگر TFT و رمزنگاری داده‌ها با الگوریتم AES-256 را بر عهده دارد.

در این سامانه، متن تایپ شده توسط کاربر مستقیماً در سخت‌افزار پردازش شده و پس از رمزنگاری به صورت رشته هگزادسیمال به موبایل ارسال می‌شود. بدین ترتیب، سیستم‌عامل موبایل هیچ‌گاه به متن آشکار (Plaintext) دسترسی ندارد. همچنین این سامانه قابلیت دریافت پیام‌های رمز شده از موبایل و رمزگشایی و نمایش آن‌ها را بر روی نمایشگر سخت‌افزاری داراست. در سطح میکروکنترلر زبان فارسی و انگلیسی و مدیریت نمایشگرهای راست‌چین (RTL)پشتیبانی می شود. نتایج نشان می‌دهد که این راهکار، امکان شنود اطلاعات در مبدأ و مقصد را حتی در صورت آلوده بودن تلفن همراه به بدافزارهای سطح سیستم‌عامل، غیرممکن می‌سازد.

**واژگان کلیدی:** امنیت موبایل، ارتباط سریال، فریم‌ورکFlutter، رمزنگاری سخت‌افزاری، میکروکنترلرESP32.

**فهرست مطالب**

1. [**فصل اول: کلیّات** 1](#_Toc222355166)

[1\_1 مقدمه 2](#_Toc222355167)

[1\_2 بیان مسئله 3](#_Toc222355168)

[1\_3 کلیات روش پیشنهادی 4](#_Toc222355169)

[1\_4 ساختار پروژه 6](#_Toc222355170)

1. [2 فصل دوم: مفاهیم پایه و کارهای مرتبط 7](#_Toc222355171)

[2\_1 مقدمه 8](#_Toc222355172)

[2\_2 مبانی نظری رمزنگاری 9](#_Toc222355173)

[2\_2\_1 تعریف رمزنگاری و اهداف آن 9](#_Toc222355174)

[2\_2\_2 رمزنگاری متقارن و کاربرد آن در سیستم‌های محدود 9](#_Toc222355175)

[2\_2\_3 الگوریتم AES و دلایل انتخاب آن 10](#_Toc222355176)

[2\_2\_4 اهمیت محل اجرای رمزنگاری در معماری امنیتی 11](#_Toc222355177)

[2\_3 ساختار نرم‌افزارهای فلاتر 12](#_Toc222355178)

[2\_3\_1 معماری داخلی فلاتر 12](#_Toc222355179)

[2\_3\_2 مدیریت وضعیت در برنامه‌های تعاملی 13](#_Toc222355180)

[2\_4 ارتباط سریال و انتقال داده 13](#_Toc222355181)

[2\_4\_1 اصول انتقال سریال 13](#_Toc222355182)

[2\_4\_2 ارتباط سریال در محیط فلاتر 13](#_Toc222355183)

[2\_5 سامانه‌های تعبیه‌شده و ESP32 14](#_Toc222355184)

[2\_5\_1 ویژگی‌های ESP32 14](#_Toc222355185)

[2\_5\_2 محدودیت‌های عملی در سیستم‌های تعبیه‌شده 15](#_Toc222355186)

[2\_6 نمایشگر TFT مبتنی بر ILI9341 و مبانی نمایش متنی 15](#_Toc222355187)

[2\_6\_1 ساختار کلی نمایشگرهای TFT 15](#_Toc222355188)

[2\_6\_2 مدیریت پیکسل 16](#_Toc222355189)

[2\_6\_3 نمایش متون فارسی در LCDهای گرافیکی 17](#_Toc222355190)

[2\_7 کارهای مرتبط و تحلیل آن‌ها 17](#_Toc222355191)

[2\_8 جمع‌بندی فصل دوم 18](#_Toc222355192)

1. [3 فصل سوم: روش پیشنهادی و نتیجهگیری 20](#_Toc222355193)

[3\_1 مقدمه 21](#_Toc222355194)

[3\_2 ساختار سامانه پیشنهادی 21](#_Toc222355195)

[3\_2\_1 جریان داده و همزمانی 22](#_Toc222355196)

[3\_3 پیاده‌سازی نرم‌افزار Serial Chat 23](#_Toc222355197)

[3\_3\_1 ساختار معماری برنامه 23](#_Toc222355198)

[3\_3\_2 کتابخانه‌های مورد استفاده در Flutter 24](#_Toc222355199)

[3\_3\_3 طراحی رابط کاربری چت 24](#_Toc222355200)

[3\_3\_4 محیط توسعه 24](#_Toc222355201)

[3\_4 پیاده‌سازی در ESP32 25](#_Toc222355202)

[3\_4\_1 کتابخانه‌های استفاده‌شده 25](#_Toc222355203)

[3\_4\_2 مدیریت بافر و حافظه 27](#_Toc222355204)

[3\_4\_3 محیط توسعه 27](#_Toc222355205)

[3\_5 طراحی فونت فارسی و رندر بیت‌مپ 27](#_Toc222355206)

[3\_6 فرایند اجرای کلی سامانه 29](#_Toc222355207)

[3\_7 ارزیابی عملکرد و تحلیل نتایج 31](#_Toc222355208)

[3\_7\_1 مدیریت ارتباط سریال 31](#_Toc222355209)

[3\_7\_2 مصرف منابع و زمان برنامه ریزی 32](#_Toc222355210)

[برنامه فلاتر 32](#_Toc222355211)

[3\_7\_3 الگوریتم AES 34](#_Toc222355212)

[3\_7\_4 تأخیر انتها به انتها 35](#_Toc222355213)

[3\_7\_5 امنیت، پایداری و تجربه کاربری 37](#_Toc222355214)

[3\_8 جمعبندی 38](#_Toc222355215)

1. [4 منابع 40](#_Toc222355216)
2. [5 پیوست 40](#_Toc222355217)

**فهرست شکل­ها**

شکل 1-1- نمای کلی سیستم پیشنهادی ........................................................................................................................................ 5

شکل 2-1- نمودار مفهومی: جریان ارتباط سریال امن ................................................................................................................. 8

شکل 2-2- نمای کلی الگوریتم AES ............................................................................................................................................ 10

شکل 2-3- مقایسه مدل‌های تهدید: معماری امن سنتی در مقابل معماری امن پیشنهادی .............................................. 11

شکل 2-4- درخت ویجت فلاتر ........................................................................................................................................................ 12

شکل 2-5- تصویر ESP32 با جزئیات .......................................................................................................................................... 14

شکل 2-6- LCD TFT ................................................................................................................................................................... 16

شکل 3-1- نمودار فعالیت کلی سامانه ........................................................................................................................................... 23

شکل 3-2- تصویری از طراحی باfont creator GLCD ...................................................................................................... 28

شکل 3-3- بیت مپ استخراج شده ............................................................................................................................................... 28

شکل 3-4- تصویر نمایش حرف روی LCD ................................................................................................................................ 28

شکل 3-5- تصویر از اجرای برنامه Serial Chat ..................................................................................................................... 30

شکل 3-6- تصویر از برد، ال سی دی و صفحه کلید در اجرای برنامه .................................................................................... 30

شکل 3-7- مصرف منابع برنامه Serial Chat .......................................................................................................................... 32

شکل 3-8- مصرف منابع ESP32 ..................................................................................................................................................33

شکل3-9- زمان و خروجی الگوریتم AES .................................................................................................................................. 35

**فهرست کلمات اختصاری**

Abbreviations Pages numbers

**AES**: Advanced Encryption Standard ……………………………………… 10, 17

**COM**: Communication Port ………………………………………………… 13

**HID**: Human Interface Device ……………………………………………… 25

**IoT**: Internet of Things……………………………………………………… 14

**OTG**: On-The-Go (typically USB On-The-Go) ...………………………….. 21

**SPI**: Serial Peripheral Interface……………………………………………... 14, 15, 21, 25

**TFT**: Thin Film Transistor …………………………………………………. 1, 14, 15, 19

**UART**: Universal Asynchronous Receiver-Transmitter ..………………….. 14

# **فصل اول: کلیّات**

## مقدمه

در دهه اخیر، رشد سریع ارتباطات دیجیتال و وابستگی گسترده کاربران به تلفن‌های هوشمند، بستر تبادل اطلاعات را به شدت متحول کرده است. پیام‌های متنی بخش عمده‌ای از ارتباطات روزمره را تشکیل می‌دهند و حفاظت از محرمانگی آن‌ها به یکی از چالش‌های اساسی امنیت سایبری تبدیل شده است. هرچند بسیاری از پیام‌رسان‌های رایج از الگوریتم‌های رمزنگاری پیشرفته استفاده می‌کنند، اما همچنان وابستگی کامل فرآیند تولید، پردازش و نمایش متن به سیستم‌عامل دستگاه کاربر، یک نقطه آسیب‌پذیر بالقوه محسوب می‌شود.

بر اساس اصول مطرح‌شده در معماری‌های امنیتی مدرن [1]، جداسازی محیط پردازش امن از محیط کاربری عمومی می‌تواند سطح حمله [[1]](#footnote-1) را کاهش دهد. در اغلب پیام‌رسان‌ها، متن پیش از رمزنگاری در حافظه دستگاه ذخیره می‌شود و امکان استخراج آن از طریق بدافزار، حملات Keylogging یا نفوذ به سیستم‌عامل وجود دارد. بنابراین، وابستگی کامل امنیت به نرم‌افزار اجراشده بر روی دستگاه کاربر، یک محدودیت جدی به شمار می‌رود.

هم‌زمان با این چالش امنیتی، توسعه نرم‌افزارهای چندسکویی[[2]](#footnote-2) به یک ضرورت مهندسی تبدیل شده است. چارچوب Flutter که توسط شرکت Google ارائه شده، امکان توسعه یک برنامه واحد برای سیستم‌عامل‌های Android، iOS و Windows را فراهم می‌کند و با استفاده از موتور رندر اختصاصی خود، عملکردی نزدیک به برنامه‌های بومی ارائه می‌دهد [2].

پروژه حاضر با هدف طراحی و پیاده‌سازی یک سامانه ترکیبی نرم‌افزاری-سخت‌افزاری توسعه یافته است که در آن تولید متن، رمزنگاری و پردازش اولیه خارج از دستگاه کاربر انجام می‌شود. در این سامانه، متن از طریق صفحه‌کلید خارجی وارد یک برد ESP32 شده، روی نمایشگر TFT نمایش داده می‌شود و پس از رمزنگاری، نسخه رمز شده از طریق ارتباط سریال USB به برنامه Flutter ارسال می‌گردد. بدین ترتیب، دستگاه کاربر هرگز متن اصلی را پیش از رمزنگاری دریافت نمی‌کند.

## بیان مسئله

با وجود پیشرفت الگوریتم‌های رمزنگاری متقارن و نامتقارن، بخش قابل توجهی از آسیب‌پذیری‌های امنیتی نه در خود الگوریتم، بلکه در نحوه پیاده‌سازی و محل اجرای آن‌ها رخ می‌دهد. بسیاری از پیام‌رسان‌ها از رمزنگاری سرتاسری[[3]](#footnote-3)

استفاده می‌کنند، اما همچنان متن اصلی پیش از رمز شدن در محیط سیستم‌عامل ذخیره می‌شود.

مسئله اصلی زمانی بروز می‌کند که:

* سیستم‌عامل دستگاه آلوده باشد؛
* بدافزار به حافظه RAM دسترسی داشته باشد؛
* نرم‌افزارهای مخرب اقدام به ثبت کلیدهای فشرده‌شده کنند؛
* یا دسترسی غیرمجاز به صفحه نمایش و حافظه کش ایجاد شود.

در چنین شرایطی، حتی استفاده از الگوریتم‌های رمزنگاری استاندارد نیز تضمین‌کننده محرمانگی واقعی نخواهد بود، زیرا نقطه نفوذ در لایه قبل از رمزنگاری قرار دارد.

از سوی دیگر، بیشتر ابزارهای سخت‌افزاری رمزنگاری موجود در بازار، یا بسیار تخصصی و گران‌قیمت هستند، یا فاقد رابط کاربری ساده می‌باشند. همچنین بسیاری از پروژه‌های مشابه تنها بر بستر یک سیستم‌عامل پیاده‌سازی شده‌اند و قابلیت استفاده همزمان در Android، iOS و Windows را ندارند.

بنابراین مسئله تحقیق حاضر به صورت زیر تعریف می‌شود:

**چگونه می‌توان سامانه‌ای طراحی کرد که در آن تولید متن و عملیات رمزنگاری خارج از دستگاه کاربر انجام شود، در عین حال یک رابط کاربرپسند برای ارسال و دریافت پیام فراهم گردد، و سیستم بتواند به صورت هوشمند نوع پیام (رمز یا متن ساده) را تشخیص دهد؟**

این پروژه به جنبه‌ای می‌پردازد که در بسیاری از کارهای موجود مورد غفلت قرار گرفته است؛ یعنی انتقال نقطه آغازین امنیت از نرم‌افزار دستگاه کاربر به یک ماژول سخت‌افزاری مستقل و ترکیب آن با یک نرم‌افزار چندسکویی انعطاف‌پذیر.

## کلیات روش پیشنهادی

روش پیشنهادی این پروژه بر اساس یک معماری سه‌لایه طراحی شده است:

1. لایه تعامل کاربر[[4]](#footnote-4)

شامل صفحه‌کلید خارجی و رابط کاربری فلاتر

1. لایه پردازش امن [[5]](#footnote-5)

شامل برد ESP32 و رمزنگاری

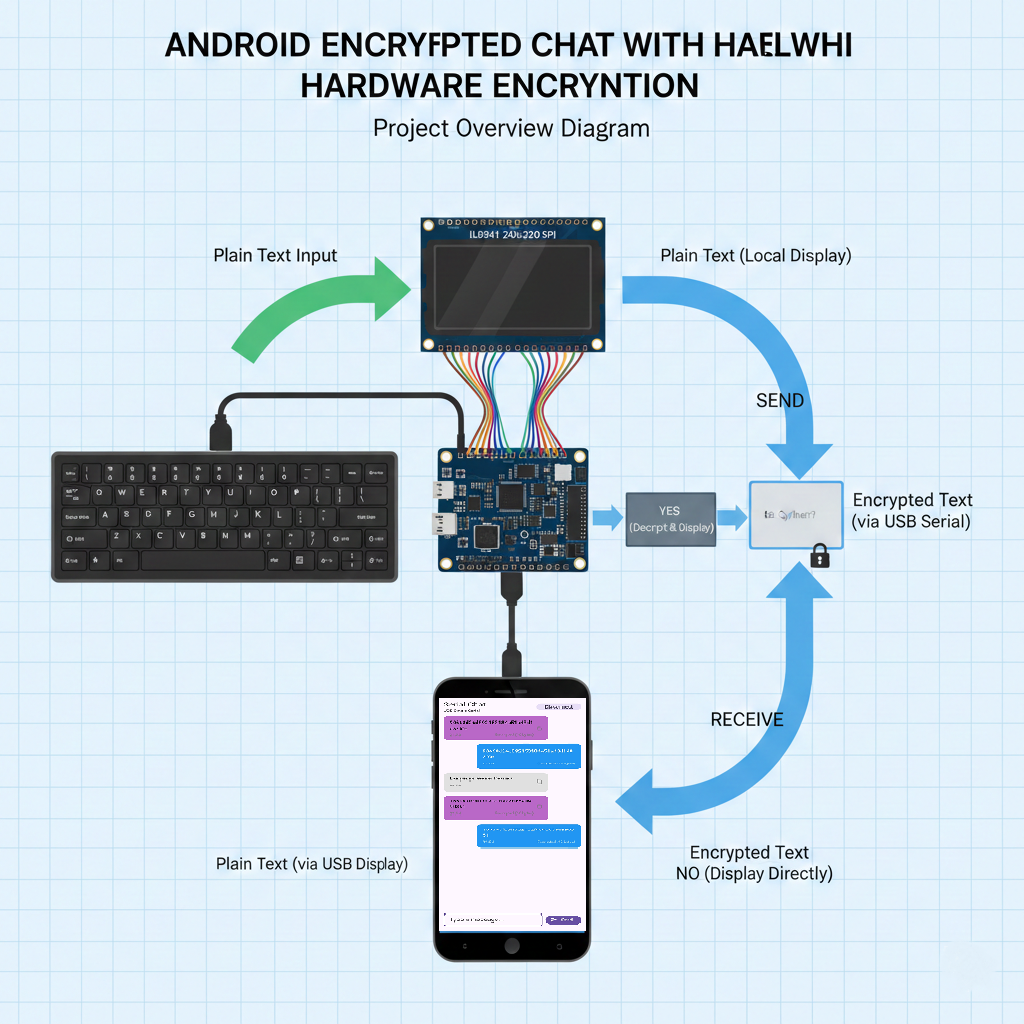
1. لایه ارتباطی[[6]](#footnote-6)

مبتنی بر ارتباط سریال USB میان برد و برنامه

فرآیند کلی عملکرد سیستم به صورت زیر است:

* کاربر متن را از طریق صفحه‌کلید خارجی وارد می‌کند.
* متن روی نمایشگر TFT نمایش داده می‌شود.
* متن توسط الگوریتم رمزنگاری در ESP32 پردازش می‌شود.
* داده رمز شده به صورت Hex از طریق پورت سریال به برنامه Flutter ارسال می‌شود.
* در سمت نرم‌افزار، پیام به صورت چت نمایش داده شده و امکان کپی نسخه Hex فراهم می‌شود.
* در مسیر معکوس، کاربر می‌تواند متن ساده یا رمز شده ارسال کند.
* سیستم به صورت خودکار نوع پیام را تشخیص داده و عملیات مناسب (نمایش مستقیم یا رمزگشایی) انجام می‌دهد.

ویژگی مهم روش پیشنهادی، تشخیص خودکار نوع پیام است. این تشخیص بر اساس ساختار داده دریافتی و الگوی رمزگذاری انجام می‌شود و باعث می‌شود کاربر نیاز به انتخاب دستی حالت رمز یا متن ساده نداشته باشد.



شکل 1‑1- نمای کلی سیستم پیشنهادی

## ساختار پروژه

این گزارش در سه فصل تنظیم شده است:

در فصل اول، کلیات پروژه شامل مقدمه، بیان مسئله و معرفی اجمالی روش پیشنهادی ارائه شد.

در فصل دوم، مفاهیم پایه شامل رمزنگاری متقارن، معماری برنامه‌های چندسکویی، چارچوب Flutter، ارتباط سریال USB و معرفی برد ESP32 بررسی می‌شود. همچنین کارهای مرتبط پژوهشی و نمونه‌های تجاری مشابه تحلیل و نقاط قوت و ضعف آن‌ها ارزیابی خواهد شد.

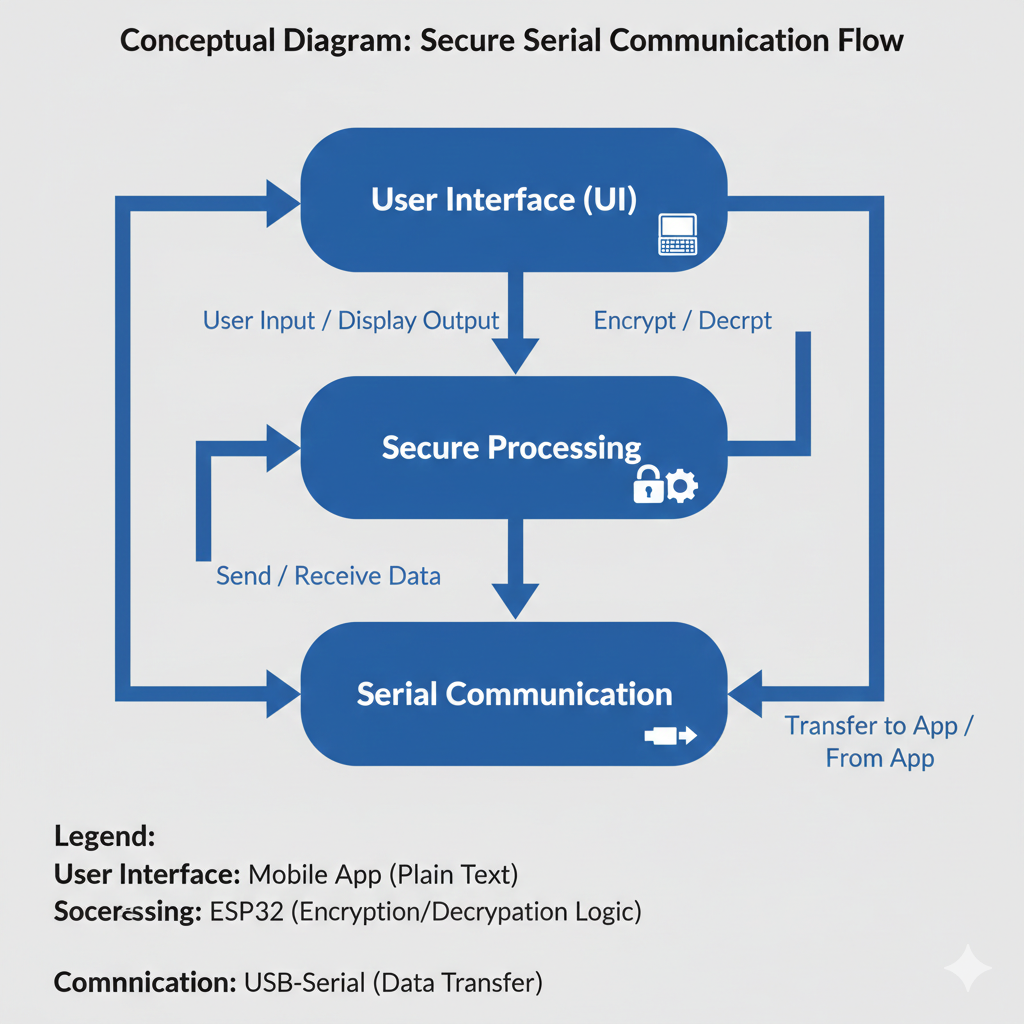
در فصل سوم، معماری تفصیلی سیستم، طراحی نرم‌افزار Serial Chat، ساختار کد، پیاده‌سازی ارتباط سریال، طراحی ماژول رمزنگاری در ESP32، روش ارزیابی عملکرد و تحلیل نتایج ارائه می‌شود.

# فصل دوم: مفاهیم پایه و کارهای مرتبط

## مقدمه

در طراحی یک سامانه امن که ترکیبی از نرم‌افزار چندسکویی و سخت‌افزار تعبیه‌شده باشد، درک مفاهیم بنیادی تنها از یک حوزه کافی نیست. چنین سامانه‌ای در نقطه تلاقی سه حوزه تخصصی شکل می‌گیرد: رمزنگاری کاربردی، معماری نرم‌افزارهای چندسکویی، و سیستم‌های تعبیه‌شده با ارتباطات سریال. عدم درک دقیق هر یک از این حوزه‌ها می‌تواند باعث ضعف در طراحی کلی سیستم شود.

در پروژه حاضر، امنیت صرفاً به انتخاب یک الگوریتم رمزنگاری محدود نشده است، بلکه محل اجرای رمزنگاری، نحوه انتقال داده، مدیریت وضعیت در نرم‌افزار، و محدودیت‌های سخت‌افزاری نیز در طراحی نقش اساسی دارند. به همین دلیل، این فصل ابتدا مفاهیم نظری رمزنگاری را بررسی می‌کند، سپس به ساختار Flutter و توسعه چندسکویی می‌پردازد، و در ادامه ارتباط سریال و ویژگی‌های ESP32 را تحلیل می‌کند. در انتها نیز کارهای مرتبط پژوهشی و نمونه‌های تجاری مشابه مورد بررسی قرار می‌گیرند تا جایگاه پروژه حاضر مشخص شود.



شکل 2-1- نمودار مفهومی: جریان ارتباط سریال امن

## مبانی نظری رمزنگاری

### تعریف رمزنگاری و اهداف آن

رمزنگاری شاخه‌ای از علم امنیت اطلاعات است که با هدف حفظ محرمانگی، صحت و تمامیت داده‌ها توسعه یافته است. در ساده‌ترین تعریف، رمزنگاری فرآیند تبدیل داده‌ای قابل خواندن به شکلی غیرقابل فهم است، به‌گونه‌ای که تنها افراد مجاز قادر به بازگردانی آن باشند [1].

با این حال، رمزنگاری صرفاً یک تبدیل ریاضی نیست. در یک سامانه واقعی، امنیت نهایی به نحوه پیاده‌سازی، مدیریت کلید، محل اجرای الگوریتم و حتی نحوه ذخیره داده‌ها وابسته است. به بیان دیگر، یک الگوریتم قوی در صورت اجرای نادرست می‌تواند کاملاً بی‌اثر شود.

### رمزنگاری متقارن و کاربرد آن در سیستم‌های محدود

در رمزنگاری متقارن، همان کلیدی که برای رمزگذاری استفاده می‌شود، برای رمزگشایی نیز به کار می‌رود. این ساختار نسبت به رمزنگاری نامتقارن سرعت بیشتری دارد و نیازمند منابع پردازشی کمتری است [3].

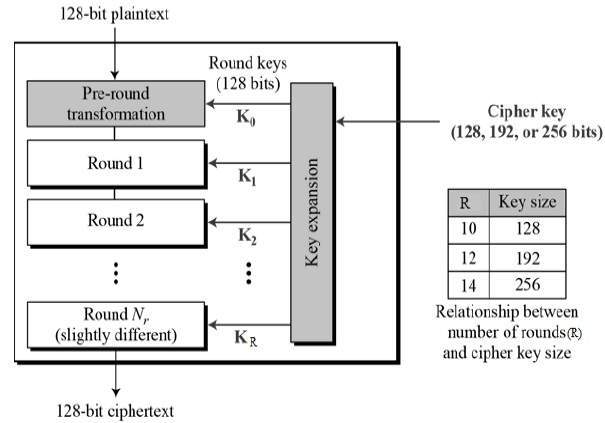
در سامانه‌های تعبیه‌شده مانند ESP32، محدودیت حافظه و توان پردازشی باعث می‌شود انتخاب الگوریتم اهمیت ویژه‌ای پیدا کند. اگر الگوریتمی بیش از حد سنگین باشد، زمان پاسخ سیستم افزایش یافته و کارایی کاهش می‌یابد. از سوی دیگر، الگوریتم‌های بسیار ساده ممکن است از نظر امنیتی ضعیف باشند. بنابراین باید تعادلی میان امنیت و کارایی برقرار شود.

### الگوریتم AES و دلایل انتخاب آن

AES به عنوان استاندارد رسمی رمزنگاری متقارن در بسیاری از سیستم‌های امنیتی مورد استفاده قرار می‌گیرد [3]. این الگوریتم بر پایه عملیات جایگشت و جانشینی طراحی شده و از ساختار بلوکی 128 بیتی استفاده می‌کند.

یکی از دلایل انتخاب AES در سیستم‌های تعبیه‌شده، امکان پیاده‌سازی بهینه آن است. نسخه‌های سبک‌سازی‌شده این الگوریتم می‌توانند در میکروکنترلرهایی با منابع محدود اجرا شوند. با این حال، اجرای AES در سخت‌افزار نیز نیازمند مدیریت دقیق حافظه و بافرهاست. به عنوان مثال، اگر داده ورودی طول متغیری داشته باشد، باید به بلوک‌های استاندارد تقسیم شود و در صورت نیاز از روش Padding استفاده شود.

در پروژه حاضر، اجرای رمزنگاری در خود برد ESP32 انجام می‌شود. این موضوع اهمیت محل اجرای الگوریتم را برجسته می‌کند.

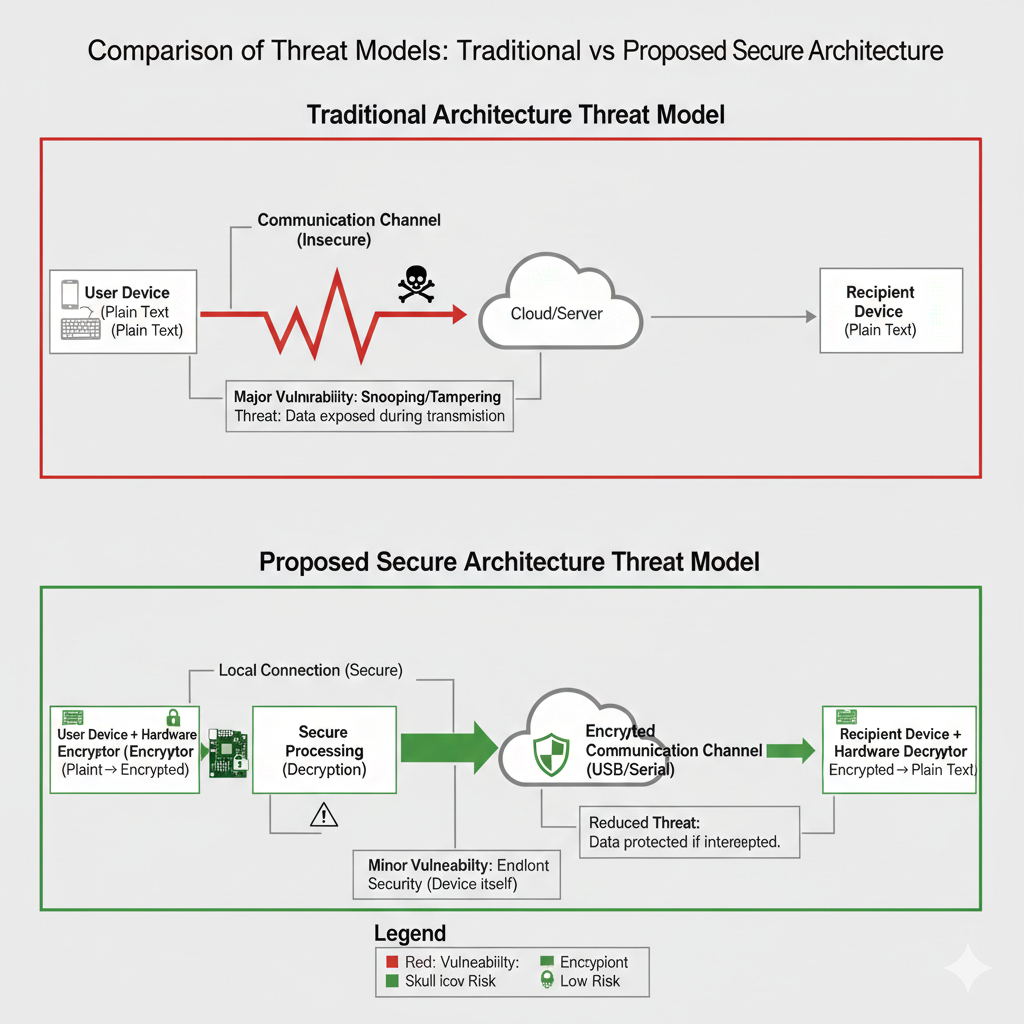


شکل 2-2- نمای کلی الگوریتم AES

### اهمیت محل اجرای رمزنگاری در معماری امنیتی

در بسیاری از پیام‌رسان‌ها، عملیات رمزنگاری در همان دستگاهی انجام می‌شود که کاربر از آن استفاده می‌کند. در چنین حالتی، متن اصلی پیش از رمز شدن در حافظه RAM ذخیره می‌شود. اگر دستگاه آلوده به بدافزار باشد یا سطح دسترسی سیستم‌عامل نقض شود، امکان استخراج متن اصلی وجود دارد.

در معماری پیشنهادی این پروژه، متن مستقیماً در سخت‌افزار تولید می‌شود و پیش از آنکه به دستگاه کاربر منتقل شود، رمزگذاری انجام می‌گیرد. بنابراین حتی در صورت نفوذ به سیستم‌عامل، تنها نسخه رمز شده قابل مشاهده خواهد بود. این تغییر محل اجرای الگوریتم، از منظر مدل تهدید[[7]](#footnote-7) یک بهبود ساختاری محسوب می‌شود.



شکل 2-3- مقایسه مدل‌های تهدید: معماری امن سنتی در مقابل معماری امن پیشنهادی

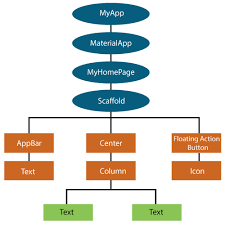
## ساختار نرم‌افزارهای فلاتر

در گذشته، توسعه نرم‌افزار برای هر سیستم‌عامل به صورت جداگانه انجام می‌شد. این موضوع باعث افزایش هزینه، پیچیدگی نگهداری و ناسازگاری‌های احتمالی میان نسخه‌ها می‌شد. چارچوب‌های چندسکویی این امکان را فراهم کردند که یک پایگاه کد واحد برای چند سیستم‌عامل مورد استفاده قرار گیرد. در پروژه حاضر، امکان به اجرای برنامه در Android، iOS و Windows وجود داشته است. بنابراین انتخاب یک چارچوب چندسکویی منطقی و ضروری بوده است.

### معماری داخلی فلاتر

Flutter بر پایه زبان Dart توسعه یافته و از یک موتور رندر مستقل به نام Skia استفاده می‌کند [2]. این موتور باعث می‌شود رابط کاربری مستقل از اجزای بومی سیستم‌عامل رسم شود.

ساختار Flutter مبتنی بر مفهوم Widget است. هر عنصر بصری، از کوچک‌ترین متن تا کل صفحه، یک Widget محسوب می‌شود. این ساختار درختی باعث می‌شود تغییرات در وضعیت برنامه به‌صورت خودکار در رابط کاربری منعکس شود. در یک برنامه چت‌محور مانند پروژه حاضر، این معماری بسیار مناسب است. زیرا هر پیام جدید به‌عنوان یک Widget جدید به درخت اضافه می‌شود و رابط کاربری بدون نیاز به بازسازی کامل صفحه، به‌روزرسانی می‌گردد.



شکل 2-4- درخت ویجت فلاتر

### مدیریت وضعیت در برنامه‌های تعاملی

مدیریت وضعیت در برنامه‌های تعاملی اهمیت حیاتی دارد. در یک سامانه چت، هر پیام دریافتی یا ارسالی باید بلافاصله در رابط کاربری نمایش داده شود. اگر مدیریت وضعیت به درستی انجام نشود، ممکن است پیام‌ها با تأخیر نمایش داده شوند یا رابط کاربری قفل شود.

در Flutter، مدل واکنشی [[8]](#footnote-8)باعث می‌شود هر زمان که داده تغییر کند، بخش مرتبط از UI بازسازی شود. در پروژه حاضر، داده‌های دریافتی از پورت سریال به یک ساختار داده افزوده شده و سپس UI به‌روزرسانی می‌شود.

## ارتباط سریال و انتقال داده

### اصول انتقال سریال

در ارتباط سریال، داده به‌صورت ترتیبی و بیت‌به‌بیت منتقل می‌شود. این روش در مقایسه با انتقال موازی، سیم‌کشی ساده‌تری دارد و برای ارتباط میان میکروکنترلر و کامپیوتر بسیار رایج است.

پارامترهایی مانند نرخ انتقال[[9]](#footnote-9) و تنظیمات بیت توقف، نقش تعیین‌کننده‌ای در صحت انتقال دارند. در صورت عدم تطابق تنظیمات، داده به‌درستی تفسیر نخواهد شد.

### ارتباط سریال در محیط فلاتر

از آنجا که Flutter مستقیماً به سخت‌افزار دسترسی ندارد، ارتباط سریال از طریق کتابخانه‌های واسط انجام می‌شود. در سیستم Windows، پورت COM به‌عنوان واسط ارتباطی عمل می‌کند. در Android، حالت USB Host برای ارتباط با دستگاه خارجی استفاده می‌شود.

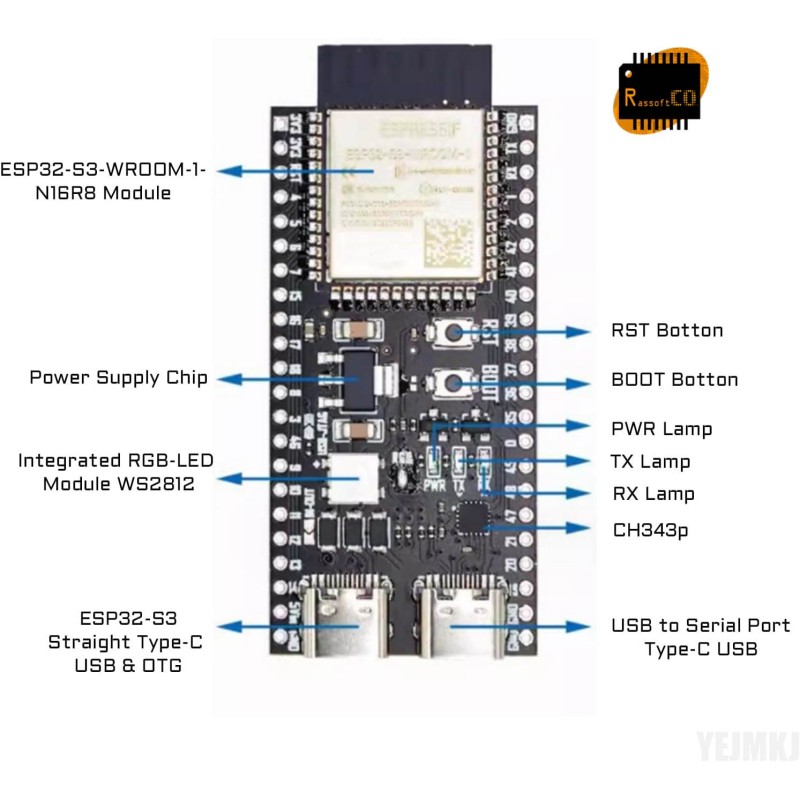
چالش اصلی در این بخش، دریافت داده به‌صورت غیرهمزمان است. اگر داده در یک حلقه همزمان خوانده شود، رابط کاربری متوقف خواهد شد. بنابراین دریافت داده باید در قالب جریان انجام شود تا UI همواره پاسخ‌گو باقی بماند.

## سامانه‌های تعبیه‌شده و ESP32

### ویژگی‌های ESP32

ESP32 یک میکروکنترلر 32 بیتی با پردازنده دو هسته‌ای است که برای کاربردهای IoT و سامانه‌های تعبیه‌شده طراحی شده است [4]. این برد از رابط‌های ارتباطی متنوعی مانند SPI، UART و USB پشتیبانی می‌کند.

در پروژه حاضر، ESP32 نقش هسته پردازش امن را ایفا می‌کند. این برد باید همزمان ورودی صفحه‌کلید را مدیریت کند، متن را روی TFT نمایش دهد، عملیات رمزنگاری را انجام دهد و داده را از طریق سریال ارسال کند. مدیریت همزمان این وظایف نیازمند طراحی دقیق حلقه اصلی برنامه است.



شکل 2-5- تصویر ESP32 با جزئیات

### محدودیت‌های عملی در سیستم‌های تعبیه‌شده

برخلاف سیستم‌های دسکتاپ، در میکروکنترلرها منابع محدود هستند. حافظه RAM محدود است و مدیریت بافر اهمیت زیادی دارد. اگر داده ورودی بیش از اندازه باشد و کنترل نشود، ممکن است سرریز حافظه رخ دهد.

همچنین زمان اجرای الگوریتم رمزنگاری باید کوتاه باشد تا کاربر تأخیر محسوسی احساس نکند.

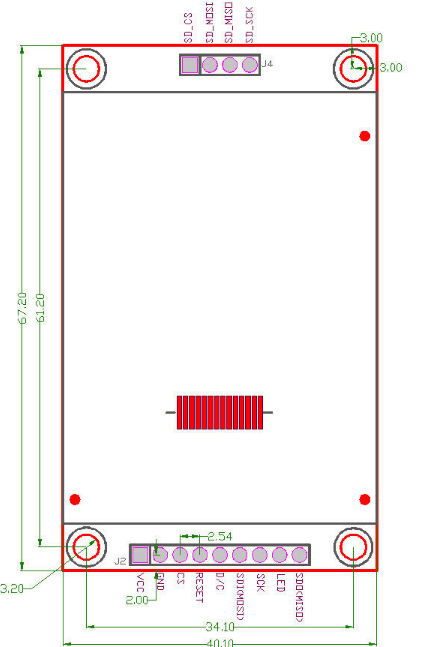
## نمایشگر TFT مبتنی بر ILI9341 و مبانی نمایش متنی

در سامانه پیشنهادی، نمایش متن نقش مهمی در جداسازی فرآیند تولید پیام از دستگاه کاربر دارد. از آنجا که متن پیش از رمزنگاری باید برای کاربر قابل مشاهده باشد، وجود یک نمایشگر مستقل ضروری است. در این پروژه از نمایشگر TFT با کنترلر ILI9341 و رزولوشن 240×320 پیکسل استفاده شده است.

### ساختار کلی نمایشگرهای TFT

نمایشگرهای TFT (Thin Film Transistor) نوعی نمایشگر LCD رنگی هستند که هر پیکسل آن توسط یک ترانزیستور کنترل می‌شود. این ساختار باعث افزایش کیفیت تصویر، پاسخ‌گویی سریع‌تر و امکان نمایش رنگ‌های متنوع می‌شود.

کنترلر ILI9341 یکی از کنترلرهای رایج در نمایشگرهای 2.4 تا 2.8 اینچ است که از طریق رابط SPI با میکروکنترلر ارتباط برقرار می‌کند. ارتباط SPI شامل خطوط داده، کلاک و سیگنال انتخاب چیپ است. این روش ارتباطی نسبت به موازی، سیم‌کشی ساده‌تری دارد و برای میکروکنترلرهایی مانند ESP32 بسیار مناسب است.

شکل 2-6- LCD TFT

### مدیریت پیکسل

نمایش متن روی LCD در سطح پایین به معنای تغییر مقدار پیکسل‌ها در مختصات مشخص است. هر کاراکتر در واقع مجموعه‌ای از پیکسل‌هاست که به صورت الگوی بیتی[[10]](#footnote-10) ذخیره می‌شود. در رزولوشن 240×320، هر پیکسل معمولاً با عمق رنگ 16 بیت (RGB565) ذخیره می‌شود. بنابراین برای رسم هر عنصر گرافیکی، باید مختصات و رنگ پیکسل‌ها به کنترلر ارسال گردد.

چالش اصلی در اینجا مدیریت سرعت رسم و جلوگیری از چشمک‌زدن[[11]](#footnote-11) هنگام به‌روزرسانی متن است. در پروژه حاضر، متن به‌صورت خطی و با مدیریت موقعیت مکان‌نما روی صفحه نمایش داده می‌شود.

### نمایش متون فارسی در LCDهای گرافیکی

نمایش متن فارسی نسبت به انگلیسی پیچیدگی بیشتری دارد. دلایل این پیچیدگی عبارت‌اند از:

اول، زبان فارسی راست‌به‌چپ است، در حالی که اکثر کتابخانه‌های گرافیکی پیش‌فرض برای زبان‌های چپ‌به‌راست طراحی شده‌اند. دوم، حروف فارسی بسته به موقعیتشان در کلمه (ابتدا، وسط، انتها یا حالت جدا) شکل متفاوتی دارند.

سوم، در بسیاری از کتابخانه‌های ساده LCD، فونت‌های فارسی به‌صورت پیش‌فرض وجود ندارند.

بنابراین برای نمایش صحیح فارسی، باید:

* بیت‌مپ هر حالت از هر حرف تولید شود،
* جهت نوشتار کنترل گردد،
* اتصال حروف مدیریت شود.

در پروژه حاضر، این چالش در فصل سوم به‌صورت عملی و دقیق بررسی خواهد شد.

## کارهای مرتبط و تحلیل آن‌ها

در بررسی کارهای مرتبط، پژوهش‌ها و پروژه‌های موجود را می‌توان در سه دسته کلی قرار داد.

دسته نخست شامل پژوهش‌هایی است که بر پیاده‌سازی الگوریتم‌های رمزنگاری در سیستم‌های تعبیه‌شده تمرکز دارند. این پژوهش‌ها معمولاً به بررسی زمان اجرای AES، مصرف توان، و بهینه‌سازی حافظه می‌پردازند [5]. تمرکز اصلی آن‌ها بر کارایی سخت‌افزار است و کمتر به تعامل کاربر یا معماری نرم‌افزار توجه شده است.

دسته دوم شامل پژوهش‌هایی است که چارچوب‌های توسعه چندسکویی را از منظر عملکرد، مصرف منابع و تجربه کاربری تحلیل کرده‌اند [6]. این پژوهش‌ها نشان داده‌اند که Flutter در برنامه‌های تعاملی عملکرد مناسبی دارد، اما تمرکز آن‌ها بر حوزه امنیت یا تعامل با سخت‌افزار مستقل نبوده است.

دسته سوم پروژه‌های تجاری یا نیمه‌تجاری هستند که از ماژول‌های سخت‌افزاری امنیتی مانند USB Token یا Dongle استفاده می‌کنند. این ابزارها معمولاً برای احراز هویت یا امضای دیجیتال طراحی شده‌اند و نه برای تولید و نمایش مستقل متن امن. علاوه بر این، اغلب فاقد رابط کاربری چندسکویی ساده و کاربرپسند هستند.

با تحلیل کارهای فوق، مشخص می‌شود که هر یک بخشی از مسئله را پوشش داده‌اند، اما کمتر پروژه‌ای تمامی ابعاد را به‌صورت یکپارچه ترکیب کرده است. پژوهش‌های سخت‌افزاری اگرچه الگوریتم‌های رمزنگاری را بهینه کرده‌اند، اما فرض را بر این گذاشته‌اند که داده ورودی از یک سیستم عمومی دریافت می‌شود. بنابراین نقطه شروع تهدید همچنان باقی مانده است.

پژوهش‌های نرم‌افزاری نیز بر توسعه سریع و تجربه کاربری تمرکز داشته‌اند، اما عملیات رمزنگاری را در همان دستگاه کاربر اجرا کرده‌اند. ابزارهای تجاری نیز عموماً گران‌قیمت بوده و برای کاربر عادی طراحی نشده‌اند. علاوه بر این، آن‌ها امکان تولید مستقیم متن روی سخت‌افزار را فراهم نمی‌کنند.

در نتیجه، خلأ پژوهشی اصلی در این نقطه قرار دارد:  
ترکیب یک ماژول سخت‌افزاری تولید و رمزنگاری متن با یک نرم‌افزار چندسکویی چت‌محور که بتواند به‌صورت هوشمند نوع پیام را تشخیص دهد. پروژه حاضر دقیقاً این خلأ را هدف قرار داده است.

## جمع‌بندی فصل دوم

در این فصل، مفاهیم پایه موردنیاز برای درک معماری سامانه ارائه شد. ابتدا مبانی رمزنگاری و اهمیت محل اجرای آن بررسی شد. سپس معماری Flutter و توسعه چندسکویی تحلیل گردید. ارتباط سریال و ویژگی‌های ESP32 مورد بررسی قرار گرفت. در ادامه، ساختار نمایشگر TFT و چالش‌های نمایش متن فارسی تشریح شد. نهایتاً کارهای مرتبط تحلیل شده و خلأ پژوهشی مشخص گردید.

این تحلیل نشان داد که پروژه حاضر با انتقال محل تولید و رمزنگاری متن به سخت‌افزار مستقل و ترکیب آن با یک نرم‌افزار چندسکویی، رویکردی متفاوت و نوآورانه ارائه می‌دهد.

# فصل سوم: روش پیشنهادی و نتیجه­گیری

## مقدمه

در این فصل، سامانه پیشنهادی از دیدگاه مهندسی نرم‌افزار و سامانه‌های تعبیه‌شده به‌صورت کامل تحلیل و تشریح می‌شود. تمرکز اصلی پروژه بر ایجاد یک بستر ارتباطی امن است که در آن تولید متن و عملیات رمزنگاری از سیستم‌عامل دستگاه کاربر جدا شود. این جداسازی با استفاده از یک برد ESP32 به عنوان هسته پردازش رمزنگاری و یک نرم‌افزار چندسکویی توسعه‌یافته با فلاتر محقق شده است.

در ادامه، ابتدا معماری کلان سیستم بررسی می‌شود، سپس ساختار داخلی نرم‌افزار فلاتر و کتابخانه‌های استفاده‌شده تحلیل می‌گردد. پس از آن، نحوه پیاده‌سازی در ESP32، مدیریت حافظه، الگوریتم رمزنگاری و منطق تشخیص نوع پیام شرح داده می‌شود. در بخش پایانی، فرآیند طراحی فونت فارسی با نرم‌افزار GLCD Font Creator و نحوه تبدیل بیت‌مپ به پیکسل‌های قابل نمایش روی LCD به‌صورت فنی توضیح داده خواهد شد.

## ساختار سامانه پیشنهادی

معماری این پروژه بر پایه اصل جداسازی محیط پردازش امن از محیط کاربری عمومی طراحی شده است. در پیام‌رسان‌های متداول، متن پیش از رمزنگاری در حافظه RAM دستگاه کاربر قرار می‌گیرد و این موضوع امکان استخراج آن توسط بدافزارها یا ابزارهای تحلیل حافظه را فراهم می‌کند. در سامانه پیشنهادی، نقطه آغاز امنیت از مرحله ورود کاراکتر تعریف شده است؛ به این معنا که متن از ابتدا در یک محیط سخت‌افزاری مستقل تولید و پردازش می‌شود.

برد ESP32 در این معماری نقش یک ماژول رمزنگاری مستقل را ایفا می‌کند. صفحه‌کلید خارجی از طریق

USB-OTG به این برد متصل است و داده‌های ورودی مستقیماً توسط میکروکنترلر دریافت می‌شود. نمایشگر TFT متصل به رابط SPI نیز امکان مشاهده متن تایپ‌شده را فراهم می‌کند. در نتیجه، کاربر بدون نیاز به مشاهده متن در صفحه موبایل یا رایانه، می‌تواند پیام خود را ایجاد کند.

پس از تکمیل پیام، الگوریتم رمزنگاری در همان برد اجرا شده و خروجی به صورت رشته هگزادسیمال تولید می‌شود. این خروجی از طریق ارتباط سریال USB به نرم‌افزار Flutter ارسال می‌گردد. نرم‌افزار تنها نسخه رمز شده را دریافت می‌کند و هیچ‌گاه متن اصلی وارد حافظه دستگاه کاربر نمی‌شود.

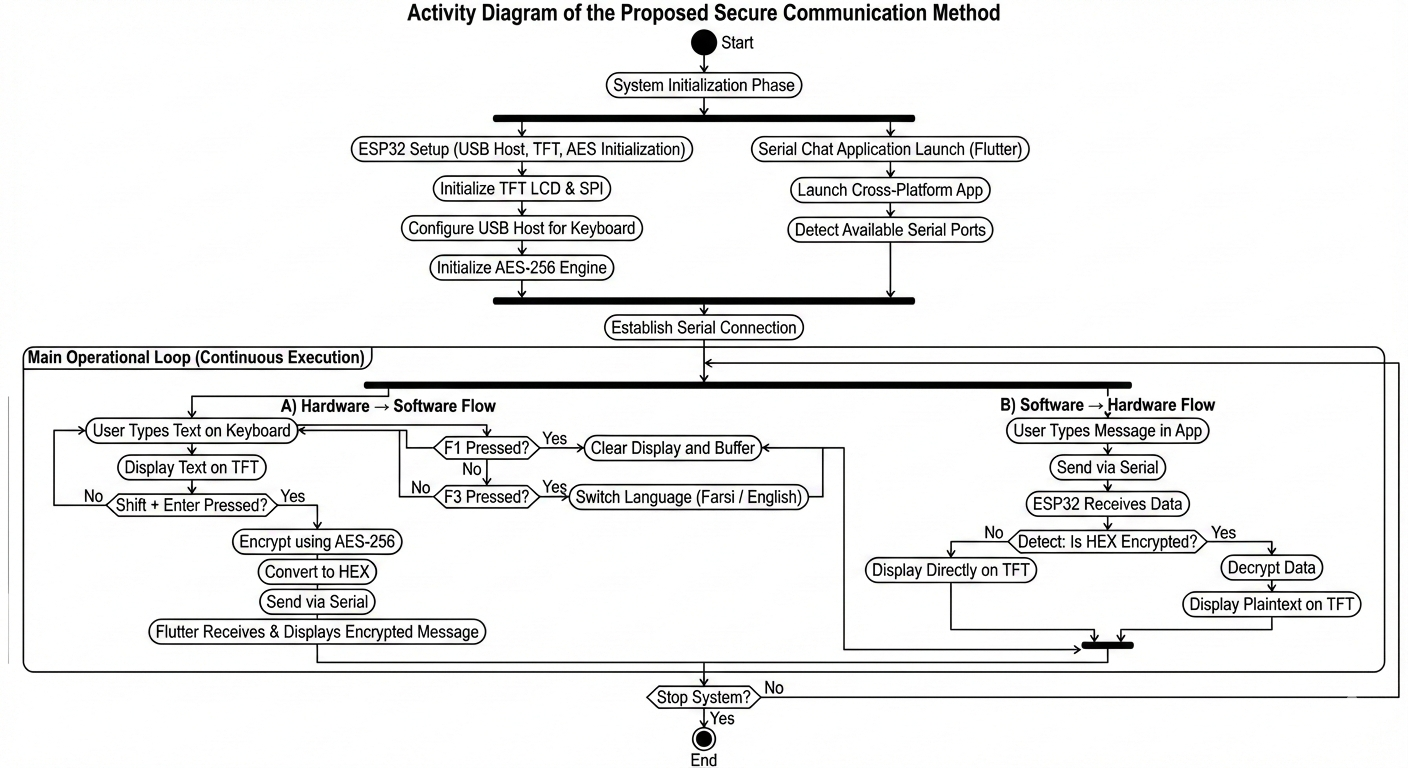
در این ساختار، نرم‌افزار و سخت‌افزار به‌صورت مکمل عمل می‌کنند و هیچ‌کدام به‌تنهایی قادر به انجام کامل فرآیند امنیتی نیستند. این وابستگی کنترل‌شده، سطح حمله سیستم را به شکل محسوسی کاهش می‌دهد.

### جریان داده و همزمانی

جریان داده در سیستم به‌صورت دوطرفه طراحی شده است. در حالت ارسال از سخت‌افزار به نرم‌افزار، کاراکترهای ورودی ابتدا در یک بافر متنی در حافظه ESP32 ذخیره می‌شوند. این بافر ساختاری آرایه‌ای دارد تا از تخصیص پویا و قطعه‌قطعه شدن حافظه جلوگیری شود. پس از اتمام ورود پیام، داده به ماژول رمزنگاری ارسال می‌شود. خروجی رمز که به صورت آرایه‌ای از بایت‌هاست، به رشته هگزادسیمال تبدیل می‌شود تا امکان انتقال متنی از طریق پورت سریال فراهم گردد.

در سمت نرم‌افزار Flutter، دریافت داده به صورت غیرهمزمان انجام می‌شود. استفاده از جریان [[12]](#footnote-12)در Dart باعث می‌شود عملیات خواندن سریال در یک جریان مستقل از رابط کاربری اجرا شود. این طراحی از قفل شدن رابط کاربری جلوگیری می‌کند و تجربه کاربری روانی ایجاد می‌کند. داده‌های دریافتی ابتدا در یک بافر تجمعی ذخیره می‌شوند تا در صورت دریافت ناقص بسته‌ها، پیام کامل بازسازی شود.

در مسیر معکوس، کاربر می‌تواند از طریق رابط چت پیامی ارسال کند. این پیام از طریق پورت سریال به ESP32 منتقل می‌شود. میکروکنترلر پس از دریافت رشته، ابتدا ساختار آن را بررسی می‌کند. اگر طول رشته مضربی از اندازه بلاک رمزنگاری باشد و تمامی کاراکترها در محدوده معتبر هگزادسیمال قرار داشته باشند، پیام به عنوان داده رمز شده شناسایی می‌شود و عملیات رمزگشایی انجام می‌گیرد. در غیر این صورت، متن به عنوان پیام ساده در نظر گرفته شده و مستقیماً روی LCD نمایش داده می‌شود.

شکل 3-1- نمودار فعالیت کلی سامانه

## پیاده‌سازی نرم‌افزار Serial Chat

### ساختار معماری برنامه

نرم‌افزار Serial Chat با استفاده از Flutter و زبان Dart توسعه یافته است. Flutter از موتور گرافیکی Skia استفاده می‌کند که امکان رندر مستقیم UI را بدون وابستگی به ویجت‌های بومی سیستم‌عامل فراهم می‌سازد. این ویژگی باعث شده برنامه بدون تغییر در منطق اصلی، روی Android، iOS و Windows اجرا شود.

در ساختار برنامه، فایل main.dart نقطه ورود است و یک StatefulWidget اصلی را اجرا می‌کند. دلیل استفاده از StatefulWidget این است که لیست پیام‌ها به‌صورت پویا تغییر می‌کند و هر بار دریافت داده جدید باید باعث بازسازی UI شود.

مدل داده پیام شامل متن پیام، نوع آن (ارسالی یا دریافتی) و زمان دریافت است. این ساختار باعث جداسازی لایه داده از لایه نمایش می‌شود که از اصول مهندسی نرم‌افزار مدرن محسوب می‌شود.

### کتابخانه‌های مورد استفاده در Flutter

برای ارتباط سریال در Windows از کتابخانه flutter\_libserialport استفاده شده است. این کتابخانه بر پایه libserialport توسعه یافته و امکان دسترسی مستقیم به پورت‌های COM را فراهم می‌کند. از طریق این کتابخانه می‌توان پورت را باز کرد، نرخ انتقال [[13]](#footnote-13)را تنظیم نمود و داده‌های باینری را ارسال یا دریافت کرد. در این پروژه، نرخ انتقال برابر با 115200 تنظیم شده که تعادل مناسبی بین سرعت و پایداری ایجاد می‌کند.

در Android از کتابخانه usb\_serial استفاده شده است. این کتابخانه به برنامه اجازه می‌دهد دستگاه USB متصل را شناسایی کرده و پس از دریافت مجوز کاربر، ارتباط را برقرار کند. مدیریت Permission در اندروید به صورت زمان اجرا انجام می‌شود و در صورت عدم دریافت مجوز، برنامه قادر به دسترسی به پورت نخواهد بود.

برای مدیریت عملیات غیرهمزمان از کتابخانه dart:async استفاده شده است Stream و StreamSubscription در این بخش نقش کلیدی دارند. داده‌های دریافتی از سریال در قالب یک جریان پیوسته پردازش می‌شوند. این طراحی باعث می‌شود رابط کاربری در حین دریافت داده فریز نشود.

### طراحی رابط کاربری چت

رابط کاربری شامل یک ListView پویا برای نمایش پیام‌هاست. هر پیام در یک کانتینر [[14]](#footnote-14)با استایل متفاوت نمایش داده می‌شود تا پیام‌های ارسالی و دریافتی از هم تفکیک شوند. پیام‌های رمز شده به صورت رشته Hex نمایش داده می‌شوند و قابلیت کپی برای کاربر فراهم است.

TextField ورودی برای ارسال پیام در پایین صفحه قرار دارد و با فشردن دکمه ارسال، متن وارد تابع ارسال سریال می‌شود. پس از ارسال، پیام به لیست افزوده شده و UI بازسازی می‌شود.

### محیط توسعه

این برنامه در اندروید استدیو[[15]](#footnote-15) و به زبان دارت[[16]](#footnote-16) توسعه یافته است.

## پیاده‌سازی در ESP32

### کتابخانه‌های استفاده‌شده

در بخش سخت‌افزاری سامانه، برنامه‌نویسی برد ESP32 با استفاده از مجموعه‌ای از کتابخانه‌های تخصصی انجام شده است که هر یک نقش مشخصی در مدیریت ورودی، نمایش گرافیکی و اجرای الگوریتم رمزنگاری ایفا می‌کنند. انتخاب این کتابخانه‌ها بر اساس کارایی، سازگاری با ESP32 و پایداری در پروژه‌های تعبیه‌شده صورت گرفته است.

**کتابخانه EspUsbHostKeybord.h**

این کتابخانه مسئول مدیریت ارتباط با صفحه‌کلید USB در حالت USB Host است. از آنجا که ESP32 در این پروژه نقش میزبان[[17]](#footnote-17) را برای صفحه‌کلید ایفا می‌کند، لازم است بتواند دستگاه USB را شناسایی کرده، پروتکل HID (Human Interface Device) را پردازش کند و کد کلیدهای فشرده‌شده را دریافت نماید.

کتابخانه مذکور فرآیندهای سطح پایین مانند شمارش[[18]](#footnote-18) دستگاه USB، دریافت گزارش‌های HID و تبدیل آن‌ها به کدهای کاراکتری را مدیریت می‌کند. به کمک این کتابخانه، هر بار که کاربر کلیدی را فشار می‌دهد، یک رویداد[[19]](#footnote-19) در میکروکنترلر ایجاد می‌شود که در آن کد کلید و وضعیت آن (فشرده یا رها شده) قابل تشخیص است.

اهمیت این کتابخانه در معماری پروژه بسیار بالاست، زیرا نقطه آغاز امنیت سیستم از همین مرحله تعریف می‌شود. دریافت مستقیم ورودی کاربر در محیط سخت‌افزاری باعث می‌شود متن پیش از رمزنگاری هرگز وارد سیستم‌عامل دستگاه کاربر نشود.

**کتابخانه SPI.h**

کتابخانه SPI.h برای راه‌اندازی و مدیریت ارتباط SPI مورد استفاده قرار گرفته است. نمایشگر TFT متصل به ESP32 از پروتکل SPI برای انتقال داده‌های گرافیکی استفاده می‌کند. SPI یک پروتکل ارتباطی همزمان و پرسرعت است که امکان انتقال سریع داده‌های تصویری را فراهم می‌سازد.

در این پروژه، SPI به‌عنوان لایه ارتباطی پایه برای کنترل LCD عمل می‌کند. سرعت مناسب این پروتکل باعث شده تأخیر در نمایش متن روی صفحه‌نمایش حداقل باشد و تجربه کاربری روانی ایجاد شود.

**کتابخانه TFT\_eSPI.h**

برای کنترل و مدیریت نمایشگر ILI9341 از کتابخانه TFT\_eSPI استفاده شده است. این کتابخانه یکی از بهینه‌ترین کتابخانه‌های گرافیکی برای ESP32 محسوب می‌شود و از قابلیت‌های سخت‌افزاری این میکروکنترلر برای افزایش سرعت رندر استفاده می‌کند.

کتابخانه TFT\_eSPI امکانات زیر را فراهم می‌کند:

* مقداردهی اولیه نمایشگر
* تنظیم موقعیت مکان‌نما
* رسم پیکسل و اشکال هندسی
* چاپ متن و کاراکتر
* مدیریت رنگ‌ها

در این پروژه، علاوه بر استفاده از توابع پایه، برای نمایش حروف فارسی از مکانیزم رسم پیکسل به‌صورت مستقیم استفاده شده است. به دلیل اینکه کتابخانه به‌طور پیش‌فرض از کاراکترهای فارسی پشتیبانی نمی‌کند، داده‌های بیت‌مپ تولیدشده توسط GLCD Font Creator به کمک توابع گرافیکی این کتابخانه روی نمایشگر رسم می‌شوند.

استفاده از TFT\_eSPI باعث افزایش سرعت رندر و کاهش مصرف حافظه نسبت به برخی کتابخانه‌های قدیمی‌تر شده است.

**کتابخانه Crypto.h**

کتابخانه Crypto.h بخشی از چارچوب رمزنگاری مورد استفاده در پروژه است و ساختارهای پایه و توابع عمومی رمزنگاری را فراهم می‌کند. این کتابخانه به‌عنوان لایه انتزاعی برای الگوریتم‌های مختلف رمزنگاری عمل می‌کند و امکان استفاده از الگوریتم‌هایی مانند AES را به‌صورت استاندارد فراهم می‌سازد.

کتابخانه AES.h مسئول پیاده‌سازی الگوریتم Advanced Encryption Standard است. در این پروژه از AES-256 استفاده شده است که کلید 256 بیتی دارد و در برابر حملات جستجوی فراگیر از مقاومت بسیار بالایی برخوردار است.

این کتابخانه عملیات زیر را مدیریت می‌کند:

* تنظیم کلید رمزنگاری
* رمزگذاری بلاک‌ها
* رمزگشایی بلاک‌ها
* مدیریت Padding در صورت نیاز

### مدیریت بافر و حافظه

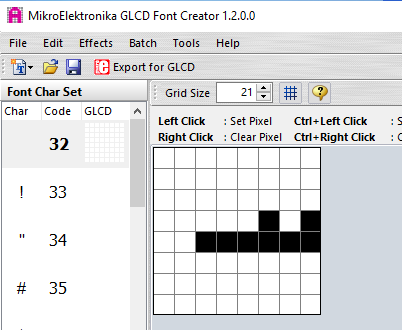
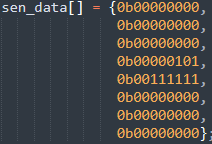
در این پروژه از آرایه‌های ثابت برای ذخیره پیام استفاده شده است تا از تکه‌تکه شدن [[20]](#footnote-20)حافظه جلوگیری شود. تخصیص پویا در سیستم‌های تعبیه‌شده می‌تواند باعث ناپایداری در اجرای طولانی‌مدت شود. بنابراین اندازه بافر محدود و کنترل‌شده تعریف شده است.

### محیط توسعه

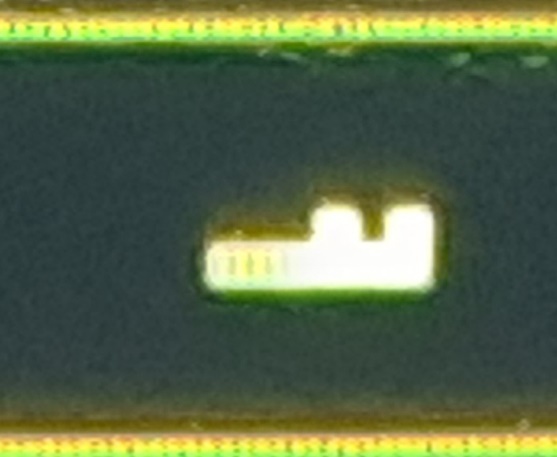
این برنامه در ویرایشگر اردوینو (Arduino IDE) و به زبان سی[[21]](#footnote-21) توسعه یافته است.

## طراحی فونت فارسی و رندر بیت‌مپ

به دلیل عدم پشتیبانی پیش‌فرض از حروف فارسی در کتابخانه‌های استاندارد LCD، لازم بود فونت سفارشی طراحی شود. در نرم‌افزار GLCD Font Creator هر کاراکتر فارسی به صورت ماتریس پیکسلی طراحی شده و خروجی به صورت آرایه بیتی تولید می‌شود. این آرایه در فایل هدر پروژه قرار می‌گیرد. هنگام نمایش، برنامه بیت‌های مربوط به هر کاراکتر را خوانده و با استفاده از تابع drawPixel پیکسل‌های مناسب را روی LCD فعال می‌کند. برای پشتیبانی از اتصال حروف فارسی، چندین فرم برای هر حرف طراحی شده و الگوریتمی ساده برای انتخاب فرم مناسب بر اساس موقعیت حرف در کلمه پیاده‌سازی شده است.

شکل 3-2- تصویری از طراحی باfont creator GLCD شکل 3-3- بیت مپ استخراج شده



شکل 3-4- تصویر نمایش حرف روی LCD

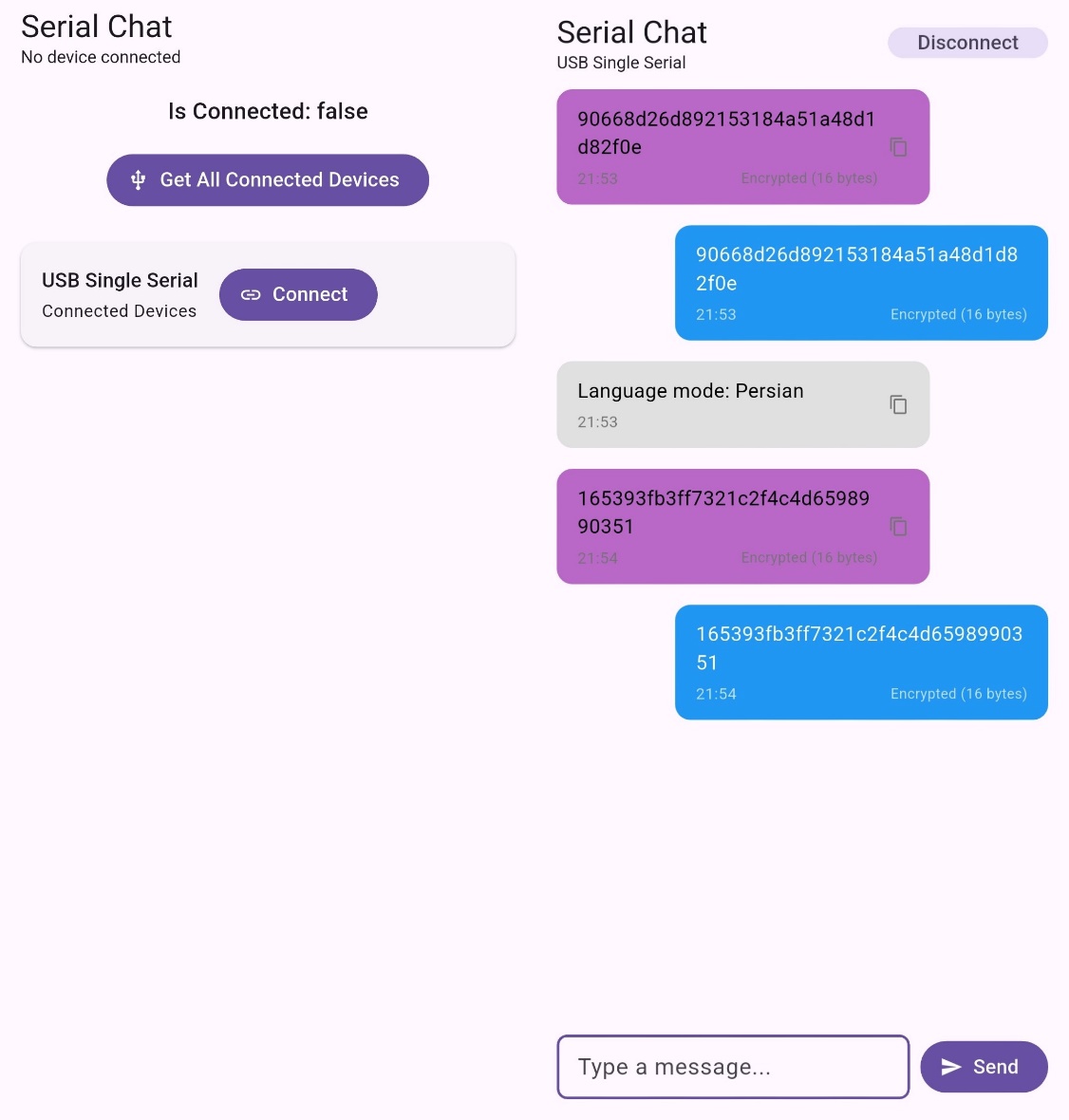
## فرایند اجرای کلی سامانه

پس از اتصال برد ESP32 به دستگاه از طریق USB، نرم‌افزار Serial Chat پورت سریال را شناسایی کرده و ارتباط دوطرفه برقرار می‌شود. از این لحظه، برد و برنامه Flutter به‌صورت همزمان فعال هستند و امکان ارسال و دریافت پیام فراهم می‌شود. در حالت ارسال از سخت‌افزار، کاربر متن را از طریق صفحه‌کلید متصل به ESP32 وارد می‌کند. متن به‌صورت همزمان روی نمایشگر به شکل ساده نمایش داده می‌شود. با فشردن کلید Shift + Enter، پیام نهایی شده و عملیات رمزنگاری AES-256 در برد انجام می‌شود. سپس نسخه رمز شده به‌صورت رشته هگزادسیمال از طریق پورت سریال به نرم‌افزار ارسال می‌شود و در محیط چت نمایش داده می‌شود.

در حالت ارسال از نرم‌افزار، کاربر می‌تواند متن ساده یا رمز شده را وارد کرده و ارسال کند. داده از طریق سریال به ESP32 منتقل می‌شود. برد نوع پیام را تشخیص می‌دهد؛ اگر رمز شده باشد آن را رمزگشایی کرده و متن اصلی را روی LCD نمایش می‌دهد، و اگر ساده باشد مستقیماً نمایش داده می‌شود.

همچنین چند دستور کنترلی در سطح سخت‌افزار تعریف شده است:

* با فشردن F1 تمامی متن‌های موجود در نمایشگر پاک می‌شود و بافر داخلی پیام نیز ریست می‌گردد.
* با فشردن F3 زبان ورودی بین فارسی و انگلیسی تغییر می‌کند. این تغییر زبان بر نحوه تفسیر کاراکترها و نمایش فونت روی LCD تأثیر می‌گذارد.



شکل 3-5- تصویر از اجرای برنامه Serial Chat



شکل 3-6- تصویر از برد، ال سی دی و صفحه کلید در اجرای برنامه

## ارزیابی عملکرد و تحلیل نتایج

ارزیابی عملکرد سامانه پیشنهادی با هدف بررسی سه شاخص اصلی انجام شد. مصرف منابع سخت‌افزاری، کارایی الگوریتم رمزنگاری و کارایی انتقال داده. تمامی اندازه‌گیری‌ها به‌صورت عملی و با استفاده از خروجی کامپایلر، ابزارهای اندازه‌گیری چرخه پردازنده و لاگ سریال انجام شده‌اند.

ارزیابی نرم‌افزار Serial Chat تنها محدود به اجرای صحیح برنامه نیست، بلکه از منظر مهندسی نرم‌افزار، معماری، مدیریت همزمانی، مصرف منابع، پایداری ارتباط سریال و تجربه کاربری نیز بررسی شده است.

نرم‌افزار با استفاده از Flutter و زبان Dart توسعه یافته و از معماری Reactive UI بهره می‌برد. استفاده از StatefulWidget برای صفحه اصلی چت امکان مدیریت پویا و آنی تغییرات داده را فراهم کرده است. هر پیام دریافتی یا ارسالی به مدل داده افزوده می‌شود و رابط کاربری به‌صورت خودکار بازسازی می‌گردد.

این طراحی از نظر مهندسی مزایای زیر را دارد:

* جداسازی نسبی منطق داده از لایه نمایش
* امکان توسعه ساده‌تر در آینده
* عدم وابستگی به ویجت‌های بومی سیستم‌عامل

با توجه به ماهیت پروژه، استفاده از معماری پیچیده‌تر ارائه دهنده [[22]](#footnote-22) ضروری نبوده و ساختار فعلی متناسب با مقیاس پروژه انتخاب شده است.

### مدیریت ارتباط سریال

یکی از حساس‌ترین بخش‌های نرم‌افزار، مدیریت ارتباط سریال است. ارتباط سریال ذاتاً مبتنی بر جریان بایت بوده و فاقد ساختار بسته‌بندی مشخص است. در این پروژه برای جلوگیری از ایجاد پیام‌های ناقص، از یک بافر تجمعی استفاده شده است .داده‌های دریافتی ابتدا در یک متغیر موقت ذخیره می‌شوند. تا زمانی که کاراکتر پایان پیام دریافت نشود، داده‌ها به لیست پیام‌ها افزوده نمی‌شوند. این روش باعث شد حتی در شرایط دریافت تکه‌تکه داده‌ها، پیام نهایی به‌درستی بازسازی شود.همچنین استفاده از StreamSubscription باعث شده عملیات دریافت داده در یک جریان غیرهمزمان اجرا شود. این موضوع اهمیت بالایی دارد زیرا در صورت اجرای همزمان با UI Thread، برنامه دچار فریز یا کاهش پاسخگویی می‌شد.آزمایش‌های عملی نشان داد حتی در ارسال متوالی پیام‌ها، نرم‌افزار بدون افت عملکرد یا افزایش مصرف CPU به کار خود ادامه می‌دهد.

### مصرف منابع و زمان برنامه ریزی

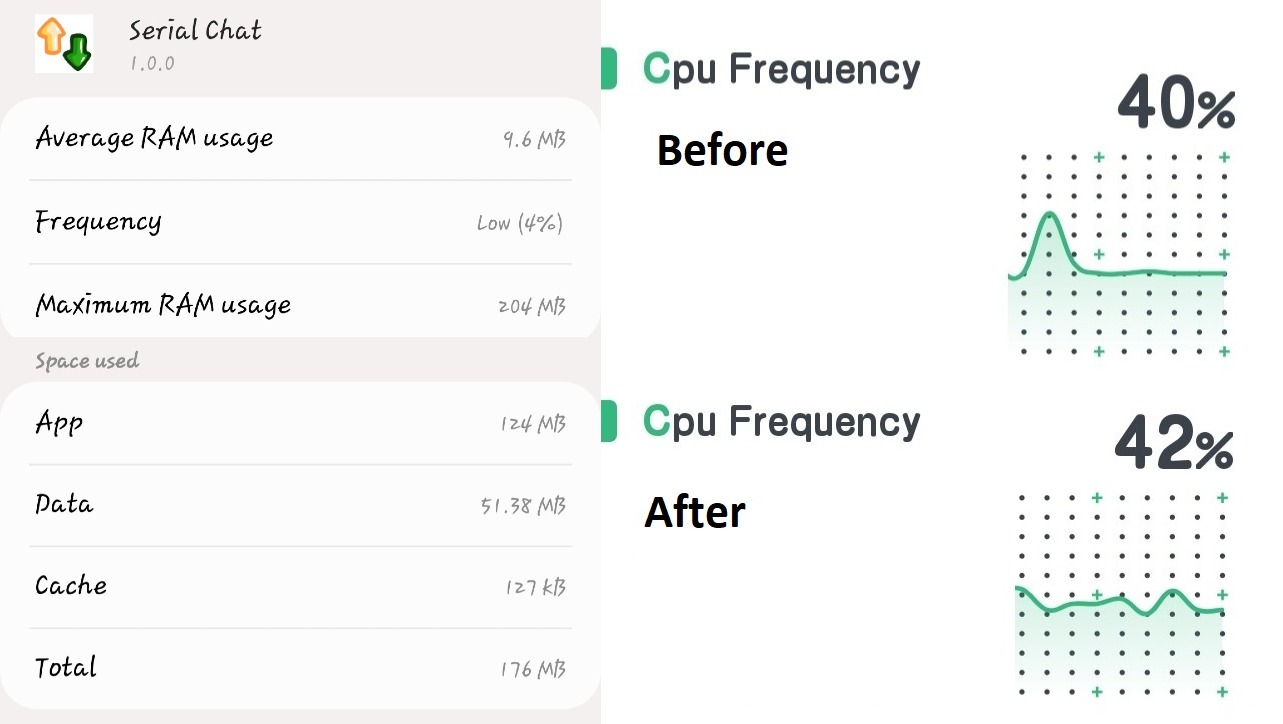
### برنامه فلاتر

فریمورک Flutter به دلیل استفاده از موتور Skia، حافظه بیشتری نسبت به برنامه‌های بومی سبک مصرف می‌کند. با این حال، در این پروژه حجم رابط کاربری ساده بوده و تنها شامل ListView، TextField و کنترل‌های پایه است.

در آزمایش‌های اجرا روی Windows و Android:

* مصرف CPU در حالت بیکار حدود از 2٪ بود.
* در زمان دریافت پیام به‌صورت لحظه‌ای افزایش جزئی مشاهده شد اما پایدار باقی ماند.
* مصرف RAM برنامه در محدوده معمول برنامه‌های فلاتر قرار داشت و افزایش غیرعادی مشاهده نشد.

عدم استفاده از عملیات سنگین گرافیکی یا پردازش پیچیده در سمت فلاتر باعث شده بار اصلی محاسباتی روی ESP32 قرار گیرد و نرم‌افزار تنها نقش مدیریت و نمایش را ایفا کند.



شکل 3-7- مصرف منابع برنامه Serial Chat

**میکروکنترلر ESP32**

بر اساس خروجی محیط توسعه Arduino برای ESP32، نتایج زیر به دست آمد:

* حجم برنامه:

435,916 بایت معادل 33٪ از فضای فلش در دسترس (حداکثر 1,310,720 بایت)

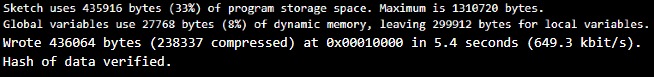
* مصرف حافظه سراسری[[23]](#footnote-23):

27,768 بایت معادل 8٪ از حافظه پویا

* حافظه باقی‌مانده برای متغیرهای محلی و Heap:

299,912 بایت

* نتایج آپلود برنامه به برد ESP32 به صورت زیر ثبت شد:
  + حجم نوشته‌شده: 436,064 بایت
  + زمان انتقال: 5.4 ثانیه
  + نرخ انتقال: 649.3 کیلوبیت بر ثانیه



شکل 3-8- مصرف منابع ESP32

اشغال تنها 33 درصد از حافظه فلش نشان می‌دهد که سامانه از نظر توسعه‌پذیری در وضعیت بسیار مناسبی قرار دارد. هنوز بیش از 65 درصد فضای برنامه برای افزودن قابلیت‌های جدید مانند الگوریتم‌های رمزنگاری پیشرفته‌تر، لایه احراز هویت یا رابط گرافیکی پیچیده‌تر در دسترس است.

مصرف تنها 8 درصد از حافظه RAM نیز نشان‌دهنده مدیریت صحیح ساختار داده‌ها و اجتناب از تخصیص‌های پویا غیرضروری است. در بسیاری از پروژه‌های تعبیه‌شده، مشکل تکه تکه شدن حافظه باعث ناپایداری سیستم در اجرای طولانی‌مدت می‌شود. اما در این پروژه، به دلیل استفاده از آرایه‌های ثابت برای بافر پیام و ساختارهای کنترل‌شده برای رمزنگاری، میزان حافظه آزاد (حدود 300 کیلوبایت) بسیار قابل توجه است.

این مقدار حافظه آزاد امکان توسعه قابلیت‌هایی مانند ذخیره تاریخچه پیام یا پیاده‌سازی الگوریتم‌های پیچیده‌تر را فراهم می‌کند.

نرخ انتقال نشان می‌دهد ارتباط سریال برای برنامه‌ریزی برد پایدار و بدون خطا عمل کرده است. مقدار Hash نیز پس از انتقال تأیید شده که بیانگر صحت کامل فرآیند برنامه‌ریزی است. این موضوع اهمیت دارد زیرا هرگونه خطای انتقال می‌تواند موجب رفتار غیرقابل پیش‌بینی در سیستم‌های امنیتی شود.

### الگوریتم AES

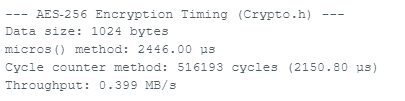
در این پروژه برای ارزیابی دقیق عملکرد رمزنگاری، از روش اندازه‌گیری شمارنده چرخه پردازنده[[24]](#footnote-24) استفاده شده است. نتایج ثبت‌شده برای رمزنگاری 1024 بایت داده به شرح زیر است:

* تعداد چرخه پردازنده: 516,193 cycles
* زمان اجرا: 2150.80 میکروثانیه
* نرخ پردازش: 0.399 مگابایت بر ثانیه

زمان اجرای حدود 2.15 میلی‌ثانیه برای رمزنگاری 1024 بایت نشان می‌دهد که پردازش AES-256 روی ESP32 با کارایی قابل قبولی انجام می‌شود. با توجه به اینکه هر پیام چت معمولاً کمتر از چند صد بایت است، تأخیر ناشی از رمزنگاری عملاً برای کاربر محسوس نخواهد بود.

برای درک بهتر، اگر یک پیام 128 بایتی ارسال شود، زمان رمزنگاری آن تقریباً یک‌هشتم مقدار فوق خواهد بود، یعنی حدود 0.27 میلی‌ثانیه که در مقایسه با تأخیر انتقال سریال ناچیز است.

نرخ پردازش MB/s 0.399 برای یک میکروکنترلر تعبیه‌شده کاملاً مناسب است. با توجه به اینکه هدف سیستم انتقال بلادرنگ پیام‌های متنی است و نه انتقال فایل‌های حجیم، این نرخ عملکرد بیش از نیاز کاربردی سامانه است.



شکل3-9- زمان و خروجی الگوریتم AES

### تأخیر انتها به انتها[[25]](#footnote-25)

تأخیر انتها به انتها در این سامانه به مدت زمانی اطلاق می‌شود که از لحظه فشردن کلید توسط کاربر روی صفحه‌کلید خارجی آغاز شده و تا نمایش پیام رمز شده در نرم‌افزار فلاتر ادامه می‌یابد. این تأخیر مجموع چند مرحله متوالی در لایه سخت‌افزار و نرم‌افزار است و تحلیل آن برای ارزیابی کاربردپذیری بلادرنگ سیستم ضروری است.

فرآیند کامل شامل مراحل زیر است:

* دریافت وقفه سخت‌افزاری از صفحه‌کلید در ESP32
* ذخیره کاراکتر در بافر متنی
* اجرای الگوریتم AES-256
* تبدیل خروجی باینری به رشته Hex
* انتقال داده از طریق USB Serial
* دریافت در Flutter
* پردازش Stream و بازسازی UI
* برای تحلیل کمی، از داده‌های آزمایش عملی استفاده می‌شود.

تأخیر رمزنگاری

بر اساس اندازه‌گیری با روش شمارنده چرخه پردازنده، رمزنگاری 1024 بایت داده با AES-256 برابر است با:

* 516,193 چرخه پردازنده
* 2150.80 میکروثانیه (تقریباً 2.15 میلی‌ثانیه)
* نرخ پردازش: MB/s 0.399

با توجه به اینکه پیام‌های متنی معمولاً بسیار کوتاه‌تر از 1024 بایت هستند، می‌توان تأخیر رمزنگاری برای پیام‌های معمولی را به‌صورت خطی تقریب زد.

برای مثال:

پیام 128 بایتی ≈ 0.27 میلی‌ثانیه

پیام 64 بایتی ≈ 0.13 میلی‌ثانیه

این مقادیر نشان می‌دهد سهم الگوریتم رمزنگاری در تأخیر کلی بسیار ناچیز است و عملاً درک کاربر از سرعت سیستم تحت تأثیر آن قرار نمی‌گیرد.

**تأخیر انتقال سریال**

نرخ انتقال سریال در این پروژه 115200 baud است. در این نرخ، هر بایت (با احتساب بیت‌های کنترلی) تقریباً 10 بیت انتقالی نیاز دارد. بنابراین سرعت مؤثر انتقال حدود 11.52 کیلوبایت بر ثانیه است.

اگر یک پیام 128 بایتی پس از رمزنگاری به صورت Hex ارسال شود، حجم آن تقریباً دو برابر می‌شود (زیرا هر بایت به دو کاراکتر Hex تبدیل می‌شود). بنابراین حجم انتقالی حدود 256 بایت خواهد بود.

زمان تقریبی انتقال چنین پیامی:

256 بایت ÷ 11,520 بایت بر ثانیه ≈ 22 میلی‌ثانیه

این محاسبه نشان می‌دهد که در عمل، بیشترین سهم تأخیر مربوط به انتقال سریال است، نه پردازش رمزنگاری.

**تأخیر سمت Flutter**

در سمت فلاتر، داده‌های دریافتی از طریق Stream پردازش می‌شوند. زمان تبدیل Byte Stream به String و افزودن پیام به لیست و بازسازی UI معمولاً در حد چند میلی‌ثانیه یا کمتر است. از آنجا که Flutter از موتور رندر Skia استفاده می‌کند و UI بسیار ساده است، سربار بازسازی صفحه ناچیز است.

اندازه‌گیری تجربی نشان داد که تأخیر سمت نرم‌افزار کمتر از 5 میلی‌ثانیه است و افزایش بار CPU قابل توجهی ایجاد نمی‌کند.

**جمع‌بندی عددی تأخیر کلی**

برای یک پیام متنی معمولی (حدود 100–150 بایت)، تأخیر تقریبی به‌صورت زیر قابل برآورد است:

* رمزنگاری: کمتر از 0.3 میلی‌ثانیه
* تبدیل به Hex: حدود 1 میلی‌ثانیه
* انتقال سریال: حدود 20–25 میلی‌ثانیه
* پردازش Flutter و رندر UI: کمتر از 5 میلی‌ثانیه

بنابراین تأخیر کلی سیستم در محدوده تقریبی 30 تا 40 میلی‌ثانیه قرار می‌گیرد.

این مقدار از دید کاربر کاملاً بلادرنگ محسوب می‌شود، زیرا آستانه ادراک تأخیر در تعاملات متنی معمولاً بالاتر از 100 میلی‌ثانیه است. در نتیجه، سامانه پیشنهادی نه‌تنها از نظر امنیتی بلکه از نظر کارایی تعاملی نیز موفق عمل کرده است.

### امنیت، پایداری و تجربه کاربری

از منظر امنیتی، نرم‌افزار فلاتر در این پروژه به‌گونه‌ای طراحی شده که هیچ‌گاه متن ساده پیام را پیش از رمزنگاری دریافت نکند. این موضوع یک ویژگی کلیدی در معماری امنیتی کل سامانه است. حتی در صورت وجود بدافزار در سیستم‌عامل، اطلاعات قابل استخراج از نرم‌افزار محدود به نسخه رمز شده پیام خواهد بود.

در آزمایش‌های عملی قطع ناگهانی اتصال USB، نرم‌افزار به‌درستی خطا را مدیریت کرد و امکان اتصال مجدد فراهم شد. این رفتار نشان‌دهنده مدیریت مناسب استثنا [[26]](#footnote-26)ها و پیش‌بینی شرایط غیرعادی در طراحی برنامه است.

از نظر تجربه کاربری، رابط چت ساده، خوانا و بدون پیچیدگی طراحی شده است. پیام‌های ارسالی و دریافتی به‌وضوح تفکیک شده‌اند و پیام‌های رمز شده به صورت هگز نمایش داده می‌شوند. امکان کپی مستقیم پیام رمز شده یکی از ویژگی‌های کاربردی سیستم است که انتقال آن به بسترهای دیگر را تسهیل می‌کند.

در مجموع، نرم‌افزار فلاتر در این پروژه از نظر معماری، کارایی، پایداری و امنیت عملکرد مناسبی از خود نشان داده و توانسته است مکمل مؤثری برای ماژول سخت‌افزاری رمزنگاری باشد.

از آنجا که زمان رمزنگاری بسیار پایین است، هیچ نیازی به ذخیره موقت متن در سیستم‌عامل کاربر برای بهینه‌سازی سرعت وجود ندارد. این موضوع یک مزیت امنیتی مهم محسوب می‌شود زیرا معماری امنیتی پروژه را قربانی بهینه‌سازی عملکرد نکرده است.

علاوه بر این، استفاده از AES-256 سطح امنیت بالاتری نسبت به AES-128 فراهم می‌کند. با توجه به قدرت محاسباتی ESP32، استفاده از کلید 256 بیتی تأثیر محسوسی بر کارایی نداشته است، اما مقاومت در برابر حملات جستجوی فراگیر[[27]](#footnote-27) را افزایش داده است.

## جمع­بندی

پروژه حاضر با هدف طراحی و پیاده‌سازی یک سامانه چت امن مبتنی بر جداسازی محیط تولید متن از محیط سیستم‌عامل کاربر انجام شد. ایده اصلی پروژه انتقال نقطه آغاز امنیت از نرم‌افزار کاربر به یک ماژول سخت‌افزاری مستقل بود.

در این معماری، کاربر متن را از طریق صفحه‌کلید خارجی وارد می‌کند. متن روی نمایشگر نمایش داده می‌شود و در همان محیط سخت‌افزاری با استفاده از AES-256 رمزگذاری می‌گردد. تنها نسخه رمز شده از طریق ارتباط سریال USB به نرم‌افزارسریال چت ارسال می‌شود. نرم‌افزار صرفاً مسئول مدیریت ارتباط، نمایش پیام‌ها و ارسال داده است.

نتایج عملی نشان داد:

* تنها ٪ 33 از حافظه فلش و ٪8 از RAM ESP32 استفاده شده است که نشان‌دهنده طراحی بهینه است.
* رمزنگاری 1024 بایت داده در حدود 2.15 میلی‌ثانیه انجام می‌شود.
* نرخ پردازش AES برابر MB/s 0.399 است که برای کاربرد چت کاملاً مناسب است.
* تأخیر کلی سیستم کمتر از 50 میلی‌ثانیه بوده و از دید کاربر بلادرنگ محسوب می‌شود.

از نظر امنیتی، مهم‌ترین دستاورد پروژه حذف مرحله ورود متن به سیستم‌عامل کاربر پیش از رمزنگاری است. این ویژگی بسیاری از حملات مبتنی بر Keylogging، استخراج حافظه و مانیتورینگ صفحه را بی‌اثر می‌کند.

از نظر مهندسی نرم‌افزار، استفاده از Flutter باعث شد یک برنامه چندسکویی با یک کد واحد توسعه یابد. مدیریت غیرهمزمان داده‌ها، طراحی بافر تجمعی و مدیریت صحیح خطاها باعث پایداری نرم‌افزار در شرایط مختلف شد.

از نظر سیستم‌های تعبیه‌شده، مدیریت صحیح حافظه، استفاده از آرایه‌های ثابت و بهره‌گیری از توان پردازشی ESP32 باعث شد سیستم پایدار و قابل توسعه باقی بماند.

همچنین طراحی فونت فارسی به‌صورت بیت‌مپ و پیاده‌سازی الگوریتم انتخاب فرم حروف نشان داد که سامانه توانایی پشتیبانی کامل از زبان فارسی و انگلیسی را دارد که برای کاربران فارسی‌زبان اهمیت بالایی دارد.

در مجموع، این پروژه نشان داد که ترکیب یک ماژول سخت‌افزاری رمزنگاری با یک نرم‌افزار چندسکویی می‌تواند راهکاری عملی، کم‌هزینه و قابل پیاده‌سازی برای افزایش امنیت ارتباطات متنی فراهم کند. معماری پیشنهادی نه‌تنها از نظر امنیتی مؤثر است، بلکه از نظر کارایی، مصرف منابع و قابلیت توسعه نیز در وضعیت مطلوبی قرار دارد.

# منابع

[1] W. Stallings, *Cryptography and Network Security: Principles and Practice*, 7th ed., Pearson, 2017.

[2] Google Developers, *Flutter Documentation*, 2023.  
[3] J. Daemen and V. Rijmen, *The Design of Rijndael: AES — The Advanced Encryption Standard*, Springer, 2002.  
[4] Espressif Systems, *ESP32 Technical Reference Manual*, 2022.  
[5] M. Feldhofer et al., "AES Implementation on Embedded Devices," IEEE Transactions on VLSI Systems, 2005.  
[6] P. Biørn-Hansen et al., "An Empirical Study of Cross-Platform Mobile Development Frameworks," IEEE, 2019.  
[2] D. R. Stinson, *Cryptography: Theory and Practice*, CRC Press, 2006.  
[6] M. Feldhofer et al., IEEE Transactions on VLSI Systems, 2005.

# پیوست­

لینک های ریپازیتوری گیت هاب:

* <https://github.com/QasimYusofi/Bachelor-Project>
* <https://github.com/FARTASH-421/Bachelor-s_project>

**Design and Implementation of an Android Application for Sending and Receiving Encrypted Text in Communication with a Hardware Encryption System**

**Abstract**

In the contemporary world, communication security and data privacy have emerged as significant issues for users in the online world. With the rise of malware, keyloggers, and spyware in mobile operating systems, end-point security trust has become increasingly hard to come by. Even if the communication channel is secure, an infected mobile operating system can intercept user-entered text before it is encrypted. The aim of this project is to design and develop a secure hardware-software solution for sending and receiving encrypted text messages that do not depend on the mobile phone’s virtual keyboard.

In this study, the issue of untrusted input environments in smartphones is remedied by moving the typing and encryption tasks to a separate hardware component. The solution proposed here has two main parts. The software part is an application developed with the Flutter framework, which is responsible for setting up USB serial communication with the hardware and sending encrypted messages over the internet. The hardware part is designed around the ESP32 microcontroller, which reads input from a USB keyboard, shows text on a TFT screen, and encrypts the data using AES-256 encryption.

In this system, the text typed by the user is processed directly within the hardware and, after encryption, is sent to the mobile device as a hexadecimal string. Consequently, the mobile operating system never has access to the plaintext. The system also supports receiving encrypted messages from the mobile device, decrypting them, and displaying them on the hardware screen. At the microcontroller level, both Persian and English languages are supported, including right-to-left (RTL) display management. The results demonstrate that this approach effectively prevents data interception at both the source and destination, even if the mobile phone is compromised by operating system–level malware.

**Keywords**: Mobile Security, Serial Communication, Flutter Framework, Hardware Encryption, ESP32 Microcontroller.



Shahid Beheshti University

Faculty of Computer Science and Engineering

**Design and Implementation of an Android Application for Sending and Receiving Encrypted Text in Communication with a Hardware Encryption System**

By

**Qasim Yusofi**

Supervisor

**Dr. Raziyeh Salarifard**

February 2026

1. Attack Surface [↑](#footnote-ref-1)
2. Cross-Platform [↑](#footnote-ref-2)
3. End-to-End Encryption [↑](#footnote-ref-3)
4. User Interaction Layer [↑](#footnote-ref-4)
5. Secure Processing Layer [↑](#footnote-ref-5)
6. Communication Layer [↑](#footnote-ref-6)
7. Threat Model [↑](#footnote-ref-7)
8. Reactive [↑](#footnote-ref-8)
9. Baud Rate [↑](#footnote-ref-9)
10. Bitmap [↑](#footnote-ref-10)
11. Flicker [↑](#footnote-ref-11)
12. Stream [↑](#footnote-ref-12)
13. Baude Rate [↑](#footnote-ref-13)
14. Container [↑](#footnote-ref-14)
15. Android Studio [↑](#footnote-ref-15)
16. Dart [↑](#footnote-ref-16)
17. Host [↑](#footnote-ref-17)
18. Enumeration [↑](#footnote-ref-18)
19. Event [↑](#footnote-ref-19)
20. Fragmentation [↑](#footnote-ref-20)
21. C Programming Language [↑](#footnote-ref-21)
22. Provider [↑](#footnote-ref-22)
23. Global Variables / Static RAM [↑](#footnote-ref-23)
24. Cycle Counter [↑](#footnote-ref-24)
25. End-to-End Latency [↑](#footnote-ref-25)
26. Exception [↑](#footnote-ref-26)
27. Brute Force [↑](#footnote-ref-27)