© February 2026 | IJIRT | Volume 12 Issue 9 | ISSN: 2349-6002

Secure Connect

Ronit Sarkar[1], Sahilkumar Basude[2], Manish Mahimkar[3], Athava Shelke[4], Prof. Rajni Ratnaparkhi[5]
[1,2,3,4,5] *Dept of Computer Engg., Dilkap College of Engineering*

Abstract—**These days, with all the cyber attacks and data leaks popping up everywhere, plus worries about who is watching our online stuff, its pretty clear we need better ways to keep communications private. I mean, secure digital privacy just feels essential now, especially in messaging.**

This project I worked on for my final year engineering is called SecureConnect. Its a web app for secure messaging that uses end to end encryption, and it has this credit system where you top up as you go. Unlike those usual apps that make you subscribe or give away your data for free tiers, here you buy credits just for what you use, like chats or calls. That gives people more say in how they spend, which I think makes sense for folks who do not message a ton, or for small businesses and stuff.

You know, sectors like healthcare or finance could really use something like this for those quick secure talks without committing to big plans. Freelancers too, probably.

On the tech side, we built the front end with React.js to make the interface easy to use on the web. For the backend, its serverless with AWS Lambda and API Gateway, which handles the processing without needing constant servers. Data gets stored encrypted in DynamoDB, including user info and credits. Authentication is through AWS Cognito, so logins stay secure.

The encryption part draws from the Signal Protocol, using things like libsodium or the WebCrypto API to keep messages private, just between the people involved. No one else can peek. Billing works by subtracting credits for each thing you do, say per message sent or minute on a call, all managed in cloud functions.

We followed agile methods to build it modularly, with ongoing deployments. It seems solid for scalability.

Sometimes I wonder if the pay per use is the best fit, some apps do freemium and it works, but others say it protects privacy more this way.

Index Terms—**Pay per use model, Encryption, Security.**

I. INTRODUCTION

In this world thats so connected all the time, with all these data breaches popping up in the news constantly, people really need ways to talk online without worrying about privacy all that much. I mean, its kind of scary how easy it is for stuff to get leaked. So something like SecureConnect comes along, and it seems like a pretty good fix for that. Its this web app for messaging, making calls, and sharing files, but the key is it keeps everything secure without costing a ton or being complicated.

The encryption part is what stands out to me. It uses end to end stuff, pulled from how apps like Signal work, so messages and voice chats stay private, no one else can peek in. That feels important, especially now.

For how it makes money, they skipped the usual subscriptions that lock you in or those free versions full of ads that get annoying. You just add credits to your wallet and pay for whatever you do, like a text or a call or sending a file. Its flexible, I think, works for people who chat a little here and there, or freelancers who dont want monthly bills, and even for jobs in places like hospitals or banks where privacy has to be super tight but without big commitments. Some might prefer unlimited plans, but this pay as you go thing makes sense for a lot of situations.

They built it on AWS, which handles the heavy lifting, like Lambda for when more people jump on, DynamoDB to keep data encrypted safely, and Cognito for logging in without issues. Its solid tech, but not over the top, just usable for regular people.

Overall, this app shows how privacy doesnt have to be a hassle, it can fit how you live. As threats keep changing online, ideas like SecureConnect help push things toward better security, bit by bit through those

safe exchanges. Not everything is perfect yet, but its a start.

II. METHODS

The whole setup for SecureConnect starts with this Agile way of building things. It focuses on making small steps forward and keeping parts separate so it can connect big ideas about digital privacy with actually using money services in real life. I think that makes sense for something like this. They use a serverless setup with microservices all on AWS to handle scaling and keeping everything secure without much hassle.

A. Frontend Architecture and Local Cryptography

The user interface is constructed using React.js, adopting a component based model for modular UI elements like the encrypted chat window and the credit management dashboard.

1. Client Side Security: Unlike traditional platforms that may process data server side, SecureConnect performs all encryption and decryption locally on the user's device.

2. Cryptographic Libraries: The system integrates libsodium (via its JavaScript wrapper) and the WebCrypto API to manage high level cryptographic primitives.

3. Key Exchange: Secure session establishment is achieved through X25519 Elliptic Curve Diffie Hellman (ECDH), ensuring that even if a future key is compromised, past communications remain protected through perfect forward secrecy.

B. Serverless backend & event driven logic

The backend operates on a serverless model to minimize operational overhead and scale automatically with user demand.

1. Compute Layer: AWS Lambda functions handle discrete business logic, such as routing encrypted "blobs" between users and managing the credit based billing triggers.

2. API Management: AWS serves as the entry point, supporting both RESTful endpoints for account management and WebSockets for real time, bidirectional communication.

3. Identity Management: AWS Cognito provides a robust authentication layer, supporting JWT based session handling and multi factor authentication without requiring a user's phone number.

C. Data Persistence & Privacy

The data layer is designed to store only what is necessary for service delivery, with an absolute focus on encryption at rest.

1. NoSQL Storage: Amazon DynamoDB is used to store user profiles and encrypted message payloads.

2. Encrypted Payloads: Messages are stored as encrypted blobs; since the private keys never leave the user's device, the database administrators cannot view the message contents.

D. Credit based Monetization System

A unique aspect of the methodology is the integration of a pay as you go billing model directly into the technical workflow.

1. Deduction Logic: Each action (sending a text, initiating a call) triggers a Lambda function that verifies the user's credit balance in DynamoDB before proceeding.

2. Usage Rates: The system applies specific credit costs: 1 credit per message, 5 credits per minute for voice calls, and 3 credits for file transfers.

3. Real Time Validation: If a user’s balance is insufficient, the system automatically disables feature access until a top up is completed through the integrated payment gateway.

III. RESULT

The SecureConnect project turned out pretty well in combining this zero knowledge setup with a pay as you go kind of money system. It kept everything encrypted end to end, even while handling credits for stuff like texts and voice chats in real time. I think using AWS serverless stuff helped a lot, since costs stayed low during the testing phase, and it showed

how this credit model could beat out those usual subscriptions.

For encryption, they used libsodium, so plain text never left the user's device. That seems solid. Benchmarks for the key exchange part say sessions start up in less than 100 milliseconds on regular browsers, and it still has that forward secrecy thing going on. Testing showed messages looked like blobs even if admins could peek at the database.

On the money side, the billing through Lambda functions deducted 1 credit per message and 5 for each voice minute. Latency was around 50 to 150 milliseconds, which fits what you expect from DynamoDB. Once the wallet hits zero, features shut off, so no free rides.

Comparing costs, for light users it's about 41 rupees a month, way cheaper than Wire or Slack, like 97 percent less. For 1000 users, infrastructure runs 12,000 to 25,000 rupees monthly, but the credits should cover that easily.

Scalability came from the microservices, no single failure point like in old apps. Cognito handled identities up to 50,000 users for free, without needing phone numbers, which makes authentication tough to break. I am not totally sure how it all ties together perfectly, but the prototype phase proved the basics work.

The monetization accuracy held up during tests. Features disabled right away at zero balance. That part stands out as reliable.

Overall, it feels like a viable alternative, though scaling to real commercial might get messy with more users.

IV. DISCUSSION

The SecureConnect project basically shows how you can keep things really secure in communication without the service provider seeing any user stuff. It uses this Zero Knowledge setup, which means encryption and all that happens right on the users device with the libsodium library. So plaintext never goes over the network or sits on servers. That part feels important because it keeps everything private from the start.

I think the billing side is interesting too. They integrated this event driven logic that works with the encryption, and it allows for pay as you go without needing to mine data or anything. Like, granular monetization but privacy stays intact. The benchmarks show the AWS serverless backend handles messaging and credit stuff with low latency. Its not like old persistent servers, this seems more efficient for real time things.

When you look at other stuff out there, like the Signal Protocol, which everyone says is top notch, SecureConnect matches it on crypto security but does better for web users. Signal relies a lot on mobile tethering from what Ive read, but this project makes a standalone web version possible. That stands out.

Also, the financial model fits with some IEEE papers on SaaS pricing. They push for usage based billing to boost customer value in clouds. Most messaging apps go freemium, but here micro transactions fund the security without messing with anonymity. I might be oversimplifying, but it works.

The main win is balancing Zero Knowledge with real time credit deductions. All the key exchanges, like X25519 ECDH, happen in the browser so the provider only sees encrypted blobs and billing metadata. Using Lambda and DynamoDB, the AWS setup acts as a blind relay. This proves privacy platforms dont have to trade off security for making money. The results confirm it runs smooth, though some parts get a bit messy in handling.

V. CONCLUSION

A. Summary of Key findings:

1. Zero Knowledge Integrity: The project successfully implemented a zero knowledge architecture, proving that end to end encryption (E2EE) can be maintained even within a serverless cloud environment. By performing cryptographic operations locally via libsodium and the WebCrypto API, the system ensures that plaintext data never reaches the server.

2. Infrastructure Scalability: Leveraging AWS serverless microservices (Lambda, DynamoDB, Cognito) allowed the platform to handle messaging and billing triggers with high performance and minimal operational overhead.

B. Future Research:

1. Cross Platform Expansion: A key priority is the development of native mobile applications (iOS and Android) to provide a seamless experience outside of the web browser, ensuring persistent connectivity and improved push notifications.

2. Advanced Encrypted Media Handling: Future iterations could incorporate more sophisticated media storage and sharing features with full client side encrypted file handling beyond basic document transfers.

3. AI Driven Security Analytics: Implementing client side machine learning to detect phishing attempts or unusual communication patterns without ever seeing the message content represents a significant frontier for preserving user safety in a zero knowledge environment.

4. Media Handling: For file sharing, the system will uses Amazon S3 with client side encryption, where files are encrypted before upload and accessed via secure, time limited signed URLs.

REFERENCES

[1] ACM Digital Library. Modern Encryption and Messaging. 2019, dl.acm.org/doi/book/10.5555/1206501.

[2] AWS Documentation. Using AWS Lambda with Amazon DynamoDB Streams. 2025, docs.aws.amazon.com/lambda/latest/dg/with ddb example.html.

[3] Blaise, Ohwo Onome. "An Understanding and Perspectives of End To End Encryption." International Research Journal, vol. 8, no. 4, 2021, www.researchgate.net/publication/350850077_ An_Understanding_and_Perspectives_of_End To End_Encryption.

[4] Gupta, Sandeep, and Bruno Crispo. "End to End Encryption for Securing Communications in Industry 4.0." 2022 4th IEEE Middle East & North African Communication Conference, IEEE, 2022, pure.qub.ac.uk/en/publications/end to end encryption for securing communications in industry 40/.

[5] IEEE Software Journal. Pay as You Go SaaS Pricing Models. IEEE Xplore, 2016, ieeexplore.ieee.org/document/7427835/.

[6] Libsodium Documentation. A Modern and Easy to Use Cryptography Library. doc.libsodium.org/.

[7] Marlinspike, Moxie. "Introduction to the Signal Protocol." WIRED, July 2016, www.wired.com/2016/07/meet moxie marlinspike anarchist bringing encryption us/.

[8] Perrin, Trevor, and Moxie Marlinspike. The Signal Protocol: Double Ratchet Algorithm. Open Whisper Systems GitHub, 2013, github.com/signalapp/libsignal protocol javascript.