Design and Development of a Password Manager With Multi-Layer Encryption Built With Rust and TypeScript
[Rancang Bangun Password Manager Dengan Enkripsi Multi-Layer Berbasis Rust dan Typescript]

Moch Jimmy Marchel [*,1)], Hamzah Setiawan [2)], Sumarno [3)], Ade Eviyanti [4)]

[1)] Program Studi Informatika, Universitas Muhammadiyah Sidoarjo, Indonesia
[2)] Program Studi Informatika, Universitas Muhammadiyah Sidoarjo, Indonesia
[3)] Program Studi Informatika, Universitas Muhammadiyah Sidoarjo, Indonesia
[4)] Program Studi Informatika, Universitas Muhammadiyah Sidoarjo, Indonesia
*Email Penulis Korespondensi: hamzah@umsida.ac.id

Abstract. *The increasing reliance of society on digital services amplifies the risk to personal data security due to the use of weak passwords. Large-scale data breach incidents prove that traditional authentication is highly vulnerable to brute-force attacks. Although password manager applications are available, some still have security vulnerabilities in memory storage and suboptimal server-side protection. This research proposes the design and development of a web-based password manager application that implements Zero-Knowledge principles with a multi-layer encryption scheme. The system is developed using Rust and TypeScript programming languages. The implemented cryptographic approach utilizes the Argon2id algorithm for the key derivation process, which is resistant to GPU brute-force attacks, and XChaCha20-Poly1305 for symmetric data encryption. All cryptographic computations are performed entirely on the client side using a Rust WebAssembly (WASM) module. This design ensures that only the ciphertext, nonce, and salt are sent to the database, while the Master Password and encryption keys never leave the user's local device memory. System testing using the Black Box Testing method demonstrates that all main features, including registration, login, and vault management, function properly in accordance with expected security standards. The research results prove that the system is capable of providing strong credential protection against data exploitation and guarantees absolute confidentiality for its users.*

Keywords - *Password Manager; Argon2; ChaCha20-Poly1305; Rust; Multi-Layer Encryption;*

Abstrak. *Peningkatan ketergantungan masyarakat terhadap layanan digital memperbesar risiko keamanan data pribadi akibat penggunaan kata sandi yang lemah. Insiden kebocoran data berskala besar membuktikan bahwa autentikasi tradisional sangat rentan terhadap serangan brute-force. Meskipun aplikasi pengelola kata sandi telah tersedia, beberapa di antaranya masih memiliki kelemahan keamanan dalam penyimpanan memori dan perlindungan sisi server yang belum optimal. Penelitian ini mengusulkan perancangan dan pengembangan aplikasi pengelola kata sandi berbasis web yang mengimplementasikan prinsip Zero-Knowledge dengan skema enkripsi multi-lapisan. Sistem dikembangkan menggunakan bahasa pemrograman Rust dan TypeScript. Pendekatan kriptografi yang diimplementasikan adalah algoritma Argon2id untuk proses derivasi kunci yang tahan terhadap serangan brute-force GPU, dan XChaCha20-Poly1305 untuk enkripsi data simetris. Seluruh komputasi kriptografi dilakukan sepenuhnya di sisi klien menggunakan modul Rust WebAssembly (WASM). Desain ini memastikan bahwa hanya ciphertext, nonce, dan salt yang dikirim ke basis data, sementara Kata Sandi Utama (Master Password) dan kunci enkripsi tidak pernah meninggalkan memori perangkat local pengguna. Pengujian sistem menggunakan metode Black Box Testing menunjukkan bahwa seluruh fitur utama termasuk registrasi, login, dan manajemen brankas berfungsi dengan baik sesuai standar keamanan yang diharapkan. Hasil penelitian membuktikan bahwa sistem mampu memberikan perlindungan kredensial yang kuat terhadap eksploitasi data dan menjamin kerahasiaan mutlak bagi penggunanya.*

Kata Kunci - *Password Manager; Argon2; ChaCha20-Poly1305; Rust; Enkripsi Multi-Layer;*

I. PENDAHULUAN

Perkembangan teknologi informasi dan komunikasi telah mendorong meningkatnya penggunaan berbagai layanan digital seperti perbankan elektronik, *e-commerce*, media sosial, layanan komputasi awan, hingga aplikasi pemerintahan berbasis elektronik sehingga setiap pengguna dituntut memiliki banyak akun dengan kredensial yang berbeda, namun dalam praktiknya masih banyak pengguna yang menggunakan kata sandi yang sama atau mudah ditebak karena keterbatasan dalam mengingat banyak kombinasi kata sandi sehingga meningkatkan risiko penyalahgunaan akun apabila terjadi kebocoran data, sebagaimana ditunjukkan oleh kasus RockYou yang

mengungkap jutaan kredensial pengguna dan menunjukkan bahwa penggunaan kata sandi yang lemah masih menjadi salah satu penyebab utama terjadinya kompromi akun digital [1].

Untuk mengatasi permasalahan tersebut, aplikasi *password manager* dikembangkan sebagai solusi yang memungkinkan pengguna membuat, menyimpan, dan mengelola kata sandi yang kuat secara terpusat sehingga setiap akun dapat menggunakan kata sandi yang unik tanpa harus diingat seluruhnya oleh pengguna, namun efektivitas aplikasi *password manager* tidak hanya ditentukan oleh kelengkapan fitur yang dimiliki melainkan juga oleh rancangan arsitektur keamanan, mekanisme pengelolaan kunci kriptografi, serta kemampuan sistem dalam melindungi data sensitif selama proses autentikasi, penyimpanan, hingga dekripsi data pengguna [2][3].

Pentingnya keamanan dalam sistem informasi tidak hanya terletak pada fitur yang tersedia, tetapi juga pada kemampuan sistem untuk diaudit guna memastikan kerahasiaan, integritas, dan ketersediaan data. Audit system informasi di era digital mencakup evaluasi mendalam terhadap kontrol teknis untuk melindungi aset informasi dari ancaman siber yang terus berkembang, yang relevan dengan perancangan pengelola kata sandi ini, di mana setiap lapisan enkripsi harus memenuhi standar keamanan yang dapat dipertanggungjawabkan melalui prosedur audit teknis [4].

Meskipun audit teknis dan arsitektur keamanan sangat penting, analisis penelitian sebelumnya menunjukkan bahwa beberapa pengelola kata sandi masih memiliki celah signifikan, termasuk penggunaan SHA-256 dan AES-GCM untuk hashing dan enkripsi yang masih dianggap aman namun belum optimal untuk menyimpan data sensitive seperti kata sandi, dan belum sepenuhnya mengintegrasikan skema enkripsi multi-lapisan yang memanfaatkan algoritma kriptografi modern seperti Argon2 untuk derivasi kunci yang tahan terhadap serangan brute-force GPU dan ChaCha20-Poly1305 untuk enkripsi simetris yang efisien [5].

Penelitian mengenai keamanan *password manager* telah dilakukan dengan berbagai pendekatan, di antaranya Cherry yang mengusulkan *Secure Password Manager Governance Framework* (Berytus) untuk meningkatkan keamanan autentikasi berbasis web terhadap serangan *Cross-Site Scripting* (XSS) dan *TLS Proxy-in-the-Middle*, Hossain yang mengembangkan *password manager* sumber terbuka dengan penerapan *secure memory wipe* dan enkripsi ganda guna meminimalkan risiko kebocoran data akibat eksploitasi memori [6][7].

Penelitian ini menerapkan rancang bangun password manager berbasis web dengan enkripsi multi-layer menggunakan Argon2id sebagai fungsi derivasi kunci dan XChaCha20-Poly1305 untuk enkripsi data simetris, di mana seluruh proses kriptografi dijalankan menggunakan Rust WebAssembly di sisi peramban pengguna guna memastikan keamanan memori sekaligus mewujudkan Zero-Knowledge di mana server hanya menyimpan ciphertext, nonce, dan salt tanpa pernah mengetahui master password atau Data Encryption Key (DEK), dengan tujuan untuk merancang, mengimplementasikan, dan mengevaluasi sistem tersebut, serta melakukan pengujian menggunakan metode Black Box Testing guna menjamin keandalan seluruh fungsi autentikasi danpengelolaan brankas.

II. METODE

2.1. Landasan Teori

Subbab ini merupakan dasar konseptual yang digunakan sebagai acuan dalam perancangan dan pengembangan sistem pada penelitian ini, dengan pembahasan yang meliputi konsep *password manager*, *Zero-Knowledge Architecture*, *Multi-Layer Encryption*, Rust, WebAssembly (WASM), TypeScript, serta algoritma kriptografi yang digunakan, yaitu Argon2id dan XChaCha20-Poly1305, beserta komponen pendukung seperti *Master Password*, *Data Encryption Key* (DEK), *salt*, *nonce*, dan metode *Black Box Testing*, sehingga setiap keputusan dalam perancangan sistem memiliki dasar teoritis yang jelas dan dapat dipertanggungjawabkan secara ilmiah.

2.1.1. Password Manager

Aplikasi yang dirancang untuk membantu pengguna membuat, menyimpan, mengelola, dan mengisi kredensial autentikasi secara aman pada berbagai layanan digital. Penggunaan *password manager* memungkinkan setiap akun memiliki kata sandi yang kuat dan unik sehingga mengurangi praktik penggunaan ulang (*password reuse*) yang menjadi salah satu penyebab utama kebocoran akun. Secara umum, *password manager* bekerja dengan menyimpan seluruh kredensial pengguna di dalam sebuah *vault* yang dilindungi menggunakan satu *master password* sebagai kunci utama untuk mengakses seluruh data yang tersimpan [8].

2.1.2. Zero-Knowledge Architecture

Ini merupakan pendekatan keamanan yang dirancang agar penyedia layanan tidak memiliki pengetahuan terhadap data sensitif maupun kunci enkripsi milik pengguna. Dalam arsitektur ini, seluruh proses pembentukan kunci, enkripsi, dan dekripsi dilakukan pada sisi klien sehingga server hanya menerima dan menyimpan data dalam bentuk *ciphertext* beserta informasi pendukung seperti *salt* dan *nonce*. Dengan pendekatan tersebut, kebocoran basis data tidak secara langsung mengungkap isi data pengguna karena server tidak pernah menyimpan *master password*, *Data Encryption Key* (DEK), maupun data dalam bentuk *plaintext*. Konsep ini banyak diterapkan pada

sistem penyimpanan data sensitif karena mampu meningkatkan kerahasiaan informasi sekaligus mengurangi risiko eksploitasi ketika infrastruktur server mengalami kompromi [9].

2.1.3. Multi-Layer Encryption

Teknik pengamanan data yang menerapkan lebih dari satu lapisan kriptografi untuk meningkatkan tingkat perlindungan terhadap informasi sensitif. Berbeda dengan enkripsi tunggal yang hanya menggunakan satu proses pembentukan kunci dan satu algoritma enkripsi, pendekatan ini memisahkan fungsi derivasi kunci, pembangkitan kunci enkripsi, serta proses enkripsi data sehingga setiap lapisan memiliki peran keamanan yang berbeda. Pada penelitian ini, lapisan pertama menggunakan Argon2id untuk menghasilkan kunci dari *master password*, sedangkan lapisan berikutnya menggunakan algoritma XChaCha20-Poly1305 untuk mengenkripsi *Data Encryption Key*(DEK) dan data yang tersimpan di dalam *vault*. Pendekatan tersebut meningkatkan ketahanan sistem terhadap serangan *brute-force*, pencurian basis data, maupun akses tidak sah terhadap informasi pengguna [10].

2.1.4. Rust

Bahasa pemrograman modern yang dikembangkan dengan fokus pada keamanan memori (*memory safety*), performa tinggi, dan konkurensi tanpa *garbage collector*. Rust menerapkan mekanisme *ownership*, *borrowing*, dan *lifetimes* untuk memastikan setiap objek dalam memori dikelola secara aman sehingga mampu mencegah berbagai kerentanan seperti *buffer overflow*, *dangling pointer*, dan *data race*. Karakteristik tersebut menjadikan Rust banyak digunakan dalam pengembangan perangkat lunak yang membutuhkan tingkat keamanan tinggi, termasuk sistem operasi, komputasi kriptografi, dan aplikasi keamanan siber. Pada penelitian ini, Rust digunakan untuk mengimplementasikan seluruh proses kriptografi agar keamanan memori tetap terjaga selama proses pembentukan kunci, enkripsi, maupun dekripsi berlangsung [11].

2.1.5. Webassembly (WASM)

Merupakan format instruksi biner yang memungkinkan kode dari berbagai bahasa pemrograman, termasuk Rust, dijalankan secara langsung di dalam peramban (*browser*) dengan performa yang mendekati aplikasi native. WASM dirancang berjalan di dalam lingkungan *sandbox* sehingga memiliki tingkat keamanan yang lebih baik dibandingkan eksekusi kode secara langsung serta tidak memberikan akses bebas terhadap sistem operasi pengguna. Dalam penelitian ini, WebAssembly digunakan untuk menjalankan seluruh proses kriptografi pada sisi klien sehingga pembentukan kunci, enkripsi, dan dekripsi dilakukan langsung di perangkat pengguna tanpa mengirimkan *master password* maupun kunci enkripsi ke server. Pendekatan ini mendukung penerapan *Zero-Knowledge Architecture* karena server hanya berfungsi sebagai media penyimpanan data terenkripsi [12].

2.1.6. TypeScript

Bahasa pemrograman yang dikembangkan sebagai *superset* dari JavaScript dengan menambahkan fitur *static typing* untuk meningkatkan keandalan proses pengembangan perangkat lunak. Penerapan sistem tipe memungkinkan kesalahan pada variabel, fungsi, maupun struktur data dideteksi sejak tahap kompilasi sehingga dapat mengurangi potensi *runtime error*. Selain mempertahankan kompatibilitas penuh dengan JavaScript, TypeScript juga mendukung pengembangan aplikasi berskala besar melalui struktur kode yang lebih terorganisasi dan mudah dipelihara. Pada penelitian ini, TypeScript digunakan untuk membangun antarmuka pengguna dan mengelola komunikasi antara aplikasi web dengan modul Rust WebAssembly secara lebih aman dan terstruktur [13].

2.1.7. Master Password

Kata sandi utama (master password) adalah satu kredensial yang berfungsi sebagai lapisan pertahanan utama dan sebagai basis entropi untuk derivasi kunci enkripsi dalam sistem pengelolaan kata sandi. Kualitasnya menentukan keamanan brankas karena kata sandi utama yang lemah dapat membuka risiko peretasan terhadap seluruh data terenkripsi, sementara kata sandi utama yang panjang, unik, dan memiliki entropi tinggi membantu menjaga integritas data sensitif dari akses tidak sah [14].

2.1.8. XchaCha20-Poly1305

Sebuah skema Enkripsi Terotentikasi dengan Data Terkait (AEAD) yang menggabungkan stream cipher XChaCha20 dan kode autentikasi pesan Poly1305 untuk menjaga kerahasiaan dan integritas data. Ia menggunakan nonce 192-bit yang mengurangi risiko tabrakan nonce dalam enkripsi massal dengan kunci yang sama, dan unggul dalam lingkungan perangkat lunak karena dapat berjalan cepat pada CPU umum tanpa instruksi perangkat keras

khusus seperti AES-NI, serta lebih tahan terhadap serangan cache-timing melalui desain berbasis ADD, ROTATE, dan XOR [15].

2.1.9. Salt

Salt dalam konteks Fungsi Derivasi Kunci Berbasis Kata Sandi (PBKDF) adalah nilai acak unik yang tidak rahasia dan ditambahkan sebelum proses hashing untuk meningkatkan keamanan terhadap serangan kamus dan serangan table pelangi (rainbow table). Salt membuat kata sandi yang identik menghasilkan hash atau kunci enkripsi yang berbeda, dan membantu memastikan bahwa kunci Argon2 untuk setiap akun bersifat unik sehingga penyerang tidak dapat melakukan dekripsi massal paralel terhadap data server [16].

2.1.10. Argon2

Algoritma fungsi derivasi kunci pemenang Kompetisi Hashing Kata Sandi (PHC) 2015, yang didefinisikan dalam RFC 9106 dan dirancang sebagai fungsi memory-hard untuk meningkatkan biaya serangan berbasis GPU atau ASIC. Varian Argon2id menggabungkan karakteristik Argon2i dan Argon2d sehingga pembentukan kunci dari kata sandi utama lebih tahan terhadap serangan side-channel dan brute-force. Penelitian pada tahun 2024 di platform TechRxiv menunjukkan bahwa arsitektur enkripsi hibrida berbasis Argon2id memberikan perlindungan yang lebih stabil terhadap skenario pencurian perangkat dibandingkan metode lama yang terbatas pada CPU [17].

2.1.11. Data Encryption Key

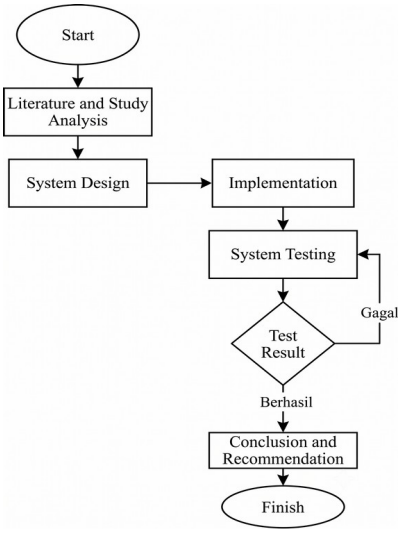
Sebuah kunci simetris acak, misalnya 256-bit, yang digunakan secara langsung untuk mengenkripsi dan mendekripsi data sensitif. Berbeda dengan kata sandi utama yang diingat pengguna, kerahasiaan ciphertext tetap terlindungi ketika basis data bocor selama DEK tidak diketahui. Dalam arsitektur yang diusulkan, DEK hanya disimpan sementara dalam memori lokal pengguna atau session storage dan dihapus ketika sesi berakhir untuk mengurangi risiko pencurian melalui memory dump [18].

2.1.12. Nonce

Nonce atau Number Used Once adalah nilai acak unik yang digunakan satu kali dalam setiap operasi enkripsi untuk memastikan ciphertext selalu berbeda meskipun data dan kunci yang digunakan sama, sehingga pola data tidak mudah dianalisis oleh penyerang dan kredensial dengan kata sandi identik pada layanan berbeda tetap menghasilkan ciphertext yang unik dalam basis data [19].

2.2. Metode Penelitian

Metode yang digunakan adalah pengembangan sistem, dengan tahapan yang disusun mulai dari identifikasi masalah, studi literatur, perancangan, implementasi, hingga pengujian keamanan seperti yang ditampilkan pada Gambar 1 (Diagram Alur Penelitian). Pendekatan ini dipilih karena penelitian berorientasi pada pembangunan dan evaluasi prototipe aplikasi, sehingga setiap persyaratan keamanan dapat langsung diterjemahkan ke dalam perancangan teknis. Setiap tahap dilakukan secara berurutan agar proses identifikasi risiko, pemilihan algoritma kriptografi, perancangan arsitektur, dan pengujian fungsional dapat dilacak dengan jelas. Dengan alur ini, hasil implementasi dapat dievaluasi berdasarkan kesesuaian antara persyaratan sistem, perancangan keamanan, dan keluaran aplikasi yang dihasilkan.



Gambar 1. Alur Penelitian

1. Identifikasi Masalah dan Tinjauan Pustaka

Tahap identifikasi masalah dilakukan untuk memetakan kerentanan pada pengelola kata sandi konvensional, khususnya risiko penyimpanan data sensitif di sisi server tanpa enkripsi end-to-end yang kuat, karena pola ini dapat meningkatkan peluang kebocoran data massal seperti kasus RockYou [20].

Studi literatur dilakukan untuk mengkaji solusi kriptografi modern yang dapat memitigasi risiko ini, dengan focus pada dua algoritma utama:

a) Argon2id sebagai Fungsi Derivasi Kunci (KDF) yang memiliki ketahanan terhadap serangan side-channel dan brute-force GPU.

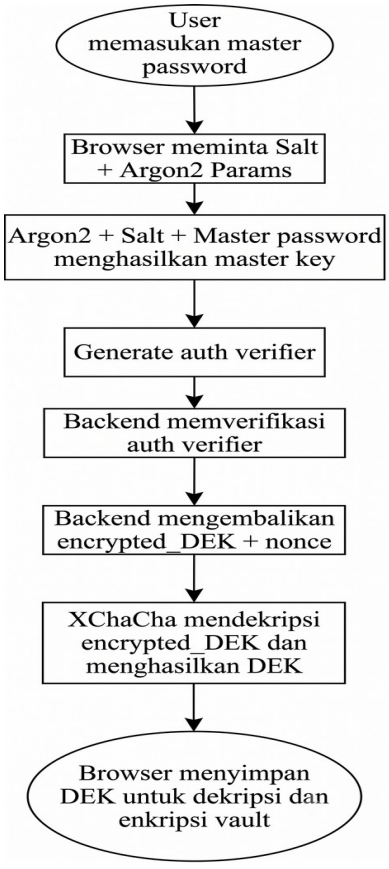
b) XChaCha20-Poly1305 sebagai algoritma enkripsi simetris yang menawarkan kinerja tinggi dan keamanan memori yang lebih baik dibandingkan AES dalam lingkungan perangkat lunak tanpa akselerasi perangkat keras.

Studi ini juga mencakup penggunaan Rust dan WebAssembly (WASM) untuk mengalihkan beban komputasi enkripsi dari server ke sisi klien, sehingga Zero-Knowledge dapat diterapkan karena server tidak pernah mengetahui kunci asli pengguna [12].

2. Perancangan Sistem

Perancangan sistem berfokus pada arsitektur keamanan aplikasi untuk menjaga integritas dan kerahasiaan data melalui skema enkripsi multi-lapisan yang berjalan sepenuhnya di browser pengguna.

Seperti yang diilustrasikan pada Gambar 2 (Diagram Arsitektur Enkripsi), sistem memisahkan zona aman (sisi klien) dan zona penyimpanan (sisi server), sementara proses pembangkitan kunci dan enkripsi data dilakukan dalam modul Rust WASM yang terisolasi sehingga data yang dikirim ke basis data hanyalah ciphertext bersama nonce dan salt, sementara kata sandi utama dan Kunci Enkripsi Data (DEK) tetap berada dalam memori lokal pengguna dan dihapus setelah sesi berakhir [21].



Gambar 2. Arsitektur Enkripsi

Diagram arsitektur enkripsi pada Gambar 2 membagi sistem menjadi dua zona utama, yaitu sisi klien (zona aman) dan sisi server (zona penyimpanan), dengan alur kerja yang terperinci sebagai berikut:

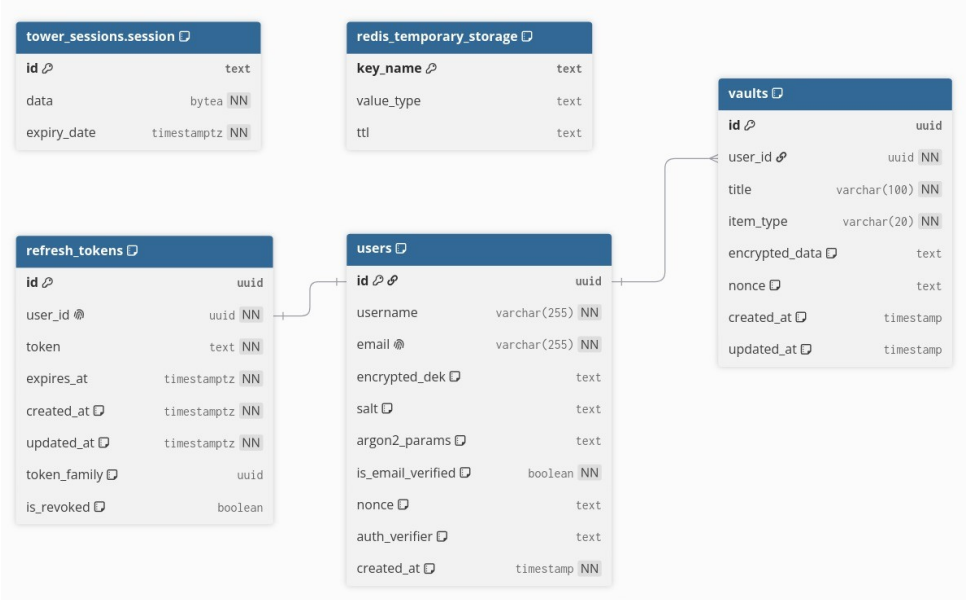
0. ***Derivasi Kunci di Sisi Klien*** **:** Proses dimulai ketika pengguna memasukkan *master password* melalui antarmuka aplikasi pada browser. Selanjutnya, modul Rust WebAssembly (WASM) mengambil salt dan

parameter Argon2id yang tersimpan pada server. Berdasarkan *master password*, *salt*, dan parameter tersebut, algoritma Argon2id dijalankan sepenuhnya di sisi klien untuk menghasilkan master key. Sesuai dengan prinsip *Zero-Knowledge Architecture*, *master password* maupun *master key* tidak pernah dikirimkan melalui jaringan ataupun disimpan pada server. Master key yang dihasilkan kemudian digunakan untuk menurunkan dua material kriptografi dengan fungsi yang berbeda, yaitu auth verifier sebagai materi autentikasi.

1. ***Proses Autentikasi* :** Setelah proses derivasi selesai, klien hanya mengirimkan auth verifier kepada server untuk proses autentikasi. Server kemudian membandingkan nilai *auth verifier* yang diterima dengan nilai yang telah tersimpan pada basis data. Apabila hasil verifikasi sesuai, server menganggap pengguna berhasil diautentikasi dan mengirimkan kembali paket data pengguna yang masih berada dalam keadaan terenkripsi, termasuk encrypted_DEK, *nonce*, serta data *vault*. Pada tahap ini, server tidak pernah menerima maupun mengetahui *master password*, *master key*, ataupun DEK dalam bentuk *plaintext*, sehingga risiko kebocoran kredensial akibat kompromi server dapat diminimalkan.
2. ***Penyediaan dan Dekripsi Data Encryption Key* :** Setelah autentikasi berhasil, klien menerima encrypted_DEK dari server. DEK tidak disimpan dalam bentuk *plaintext* pada server, melainkan terlebih dahulu dienkripsi agar hanya dapat dibuka oleh pengguna yang mengetahui *master password*. Selanjutnya, modul Rust WASM menggunakan key yang telah diturunkan pada tahap sebelumnya untuk mendekripsi *encrypted_DEK* menggunakan algoritma XChaCha20-Poly1305. Hasil proses ini berupa Data Encryption Key (DEK) dalam bentuk *plaintext* yang selanjutnya disimpan secara sementara (*volatile*) di dalam sessionStorage selama sesi pengguna masih aktif. Penyimpanan sementara tersebut memungkinkan proses enkripsi dan dekripsi dilakukan secara efisien tanpa harus menjalankan kembali proses derivasi kunci setiap kali pengguna mengakses *vault*.
3. ***Operasi Enkripsi dan Dekripsi Vault*:** DEK yang telah diperoleh kemudian digunakan sebagai kunci utama untuk seluruh proses enkripsi dan dekripsi data pada *vault*. Ketika pengguna menambahkan atau memperbarui kredensial, data akan dienkripsi secara lokal di sisi klien menggunakan algoritma XChaCha20-Poly1305, sehingga server hanya menerima ciphertext, *nonce*, dan metadata yang diperlukan untuk proses penyimpanan di PostgreSQL. Sebaliknya, ketika pengguna membuka kembali data yang tersimpan, server hanya mengirimkan ciphertext kepada klien, kemudian modul Rust WASM mendekripsinya menggunakan DEK yang berada di memori lokal hingga menghasilkan data dalam bentuk *plaintext*. Dengan mekanisme ini, seluruh proses kriptografi berlangsung sepenuhnya pada sisi klien sehingga server hanya berfungsi sebagai media penyimpanan data terenkripsi dan tidak pernah memiliki akses terhadap isi *vault* maupun kunci enkripsi milik pengguna.

2.2.1. Perancangan Basis Data

Perancangan basis data pada Gambar 3 disusun untuk mendukung Zero-Knowledge dan pemisahan data sensitif antara sisi klien dan sisi server. Struktur utama menggunakan PostgreSQL untuk penyimpanan persisten, sementara Redis digunakan sebagai penyimpanan sementara untuk OTP, batasan laju (rate limits), dan token tertentu yang memiliki masa berlaku terbatas.



Gambar 3. Perancangan Basis Data

Tabel users menyimpan identitas pengguna dan materi kriptografi yang dibutuhkan oleh klien untuk membuka brankas, seperti encrypted_dek, salt, argon2_params, nonce, dan auth_verifier. Nilai encrypted_dek merupakan DEK yang terenkripsi, sedangkan auth_verifier digunakan untuk proses verifikasi tanpa menyimpan kata sandi utama. Tabel vaults memiliki hubungan banyak-ke-satu (many-to-one) dengan users melalui user_id dan menyimpan metadata item, encrypted_data, nonce, serta waktu pembuatan dan pembaruan data brankas.

Tabel refresh_tokens digunakan untuk mengelola token autentikasi jangka panjang bersama dengan token_family dan status pencabutan token. Selain tabel-tabel utama tersebut, tower_sessions.session dibuat secara otomatis oleh session store untuk menyimpan data sesi, sedangkan Redis menyimpan data sementara seperti otp:{user_id}, rate_limit:otp, rate_limit:login, dan token:unlock dengan mekanisme TTL sehingga data pendukung tidak disimpan secara permanen.

1. Implementasi dan Pengujian Sistem

Tahap implementasi menerjemahkan desain arsitektur ke dalam kode program melalui dua komponen utama:

- Logika backend, yang merupakan implementasi fungsi kriptografi Argon2id, XChaCha20, dan Poly1305 untuk mendukung keamanan memori dan kecepatan eksekusi.
- Antarmuka frontend, yang merupakan antarmuka pengguna type-safe untuk meminimalkan kesalahan runtime saat menangani input data sensitif.

Setelah sistem dibangun, pengujian fungsional dilakukan menggunakan metode Black Box Testing untuk memverifikasi bahwa fitur utama, seperti registrasi, login, enkripsi (simpan brankas), dan dekripsi (buka brankas), berjalan sesuai spesifikasi tanpa meninjau kode internal [22].

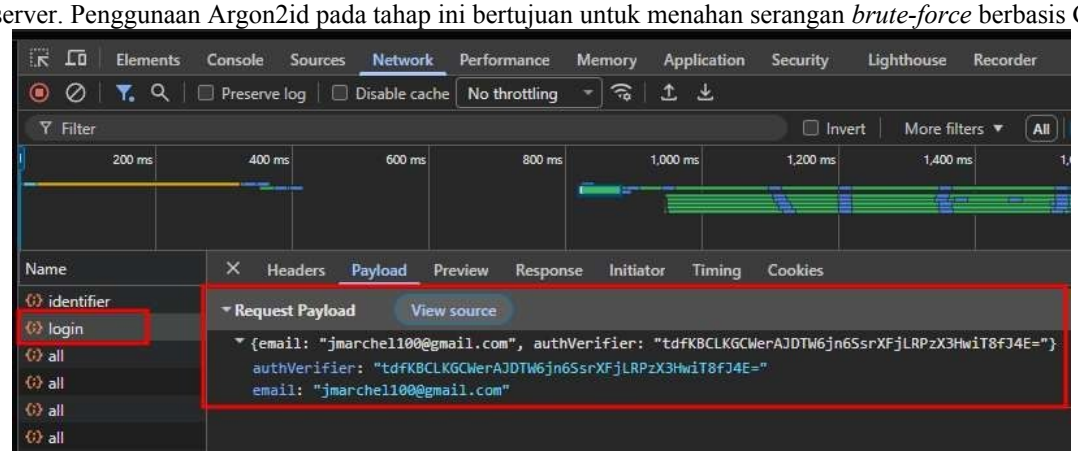
III. HASIL DAN PEMBAHASAN

3.1. Implementasi Arsitektur Enrkipsi Multi-Layer

Pada tahap implementasi ini, sistem tidak hanya berfokus pada antarmuka pengguna, melainkan pada penerapan *Zero-Knowledge* yang memisahkan tanggung jawab keamanan antara sisi klien (*browser*) dan sisi server. Seluruh proses komputasi kriptografi dieksekusi secara penuh di sisi klien menggunakan modul Rust WebAssembly (WASM). Berikut adalah penjelasan rinci mengenai alur enkripsi berdasarkan arsitektur yang telah dirancang:

1. Pemrosesan Kredensial dan Key Derivation (Sisi Klien)

Proses dimulai ketika pengguna memasukkan kredensial berupa *master password* melalui antarmuka aplikasi. Sistem tidak mengirimkan *master password* tersebut ke server, melainkan menangkapnya di memori lokal peramban (*browser*). Modul Rust WASM kemudian menjalankan algoritma **Argon2id** untuk melakukan derivasi kunci dari *master password* tersebut, menggunakan parameter dan *salt* unik yang didapatkan dari server. Penggunaan Argon2id pada tahap ini bertujuan untuk menahan serangan *brute-force* berbasis GPU.



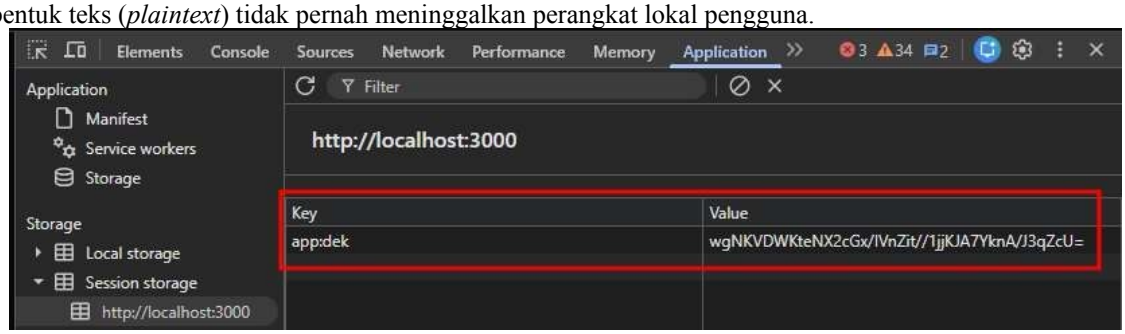
Gambar 4. Request Payload pada proses autentikasi yang menunjukan bahwa system tidak mengirimkan kata sandi.

Gambar 4 membuktikan penerapan *Zero-Knowledge*, di mana *payload* yang dikirim ke *backend* hanyalah *auth verifier* atau materi verifikasi, bukan *master password* asli yang diketikkan oleh pengguna.

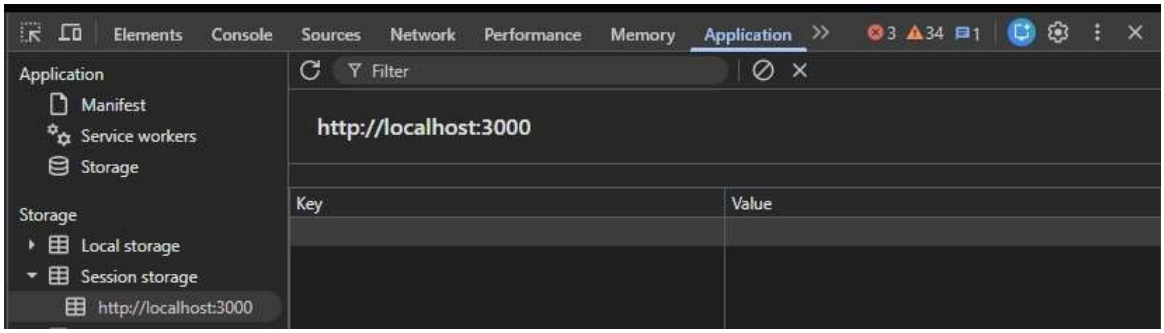
2. Verifikasi Autentikasi dan Pengambilan data (sisi server)
Hasil derivasi dari Argon2id digunakan untuk membentuk *auth verifier* yang kemudian dikirim ke server sebagai bahan verifikasi autentikasi. Jika verifier tersebut valid (langkah *Auth Verification* berhasil), server akan memberikan respons sukses dan mengirimkan paket data pengguna yang masih dalam keadaan terenkripsi. Salah satu komponen krusial yang dikembalikan oleh server ke klien adalah encrypted_DEK (Data Encryption Key yang terenkripsi).

3. Dekripsi Kunci Simetris (XChaCha20)
Setelah menerima respons dari server, modul Rust WASM di sisi klien menggunakan kunci utama (*master key*) hasil derivasi sebelumnya untuk mendekripsi encrypted_DEK. Proses dekripsi ini menggunakan algoritma **XChaCha20-Poly1305** yang efisien dijalankan pada lingkungan perangkat lunak. Hasil dari proses ini adalah sebuah kunci simetris murni (DEK) yang kini siap digunakan untuk membuka dan mengunci data-data sensitif di dalam *vault* pengguna.

4. Keamanan Sesi
DEK yang telah didekripsi hanya disimpan secara sementara (bersifat *volatile*) di dalam memori local pengguna (*session storage*) selama sesi sedang aktif. Master password, kunci enkripsi (DEK), dan isi brankas (*vault*) dalam bentuk teks (*plaintext*) tidak pernah meninggalkan perangkat lokal pengguna.



Gambar 5. Penyimpanan DEK pada Session Storage



Gambar 6. Penghapusan Otomatis saat Sesi Berakhir

Gambar 5 dan 6 membuktikan mitigasi terhadap eksploitasi memori. DEK disimpan sementara di sessionStorage untuk mempercepat proses enkripsi/dekripsi secara dinamis, dan sistem secara otomatis menghapus (menyapu bersih) data tersebut ketika pengguna menekan tombol *logout* atau sesi berakhir.

5. Penyimpanan Data Aman di Database
Ketika pengguna menambahkan atau mengubah data kredensial di dalam aplikasinya, DEK di memori akan mengenkripsi data tersebut menjadi *ciphertext*. Sistem kemudian hanya mengirimkan *ciphertext*, *nonce*, dan *salt* ke *database*.

Gambar 7. Penyimpanan Data di dalam database PostgreSQL

Gambar 7 memberikan bukti visual bahwa *database* pada sisi *backend* murni berfungsi sebagai ruang penyimpanan (*storage zone*). Kolom-kolom sensitif seperti encrypted_dek, auth_verifier, dan isi data *vault* sepenuhnya berisi deretan karakter acak (*ciphertext*). Hal ini memastikan jaminan kerahasiaan absolut; jika terjadi kebocoran basis data (*data breach*), peretas tetap tidak dapat membaca informasi tersebut karena kunci dekripsinya tidak ada di server.

3.2. Pengujian Sistem

Pengujian Black Box berfokus pada kesesuaian input dan output dari perspektif pengguna tanpa meninjau kode sistem internal. Tujuan pengujian ini adalah untuk memastikan bahwa setiap fitur berjalan sesuai skenario yang dirancang, dan hasil pengujian fitur utama disajikan dalam Tabel 1 [23].

Tabel 1. Hasil Pengujian Black Box

Skenario Pengujian	Deskripsi Skenario (Fokus Kriptografi)	Hasil yang DIharapkan	Status
Login	Menguji transmisi data saat pengguna melakukan proses *login* dengan *master password*.	Sistem mengeksekusi Argon2id di sisi klien. Pemeriksaan pada *network payload* membuktikan bahwa *password* asli tidak dikirim ke server, melainkan hanya *auth verifier*.	Valid
Derivasi DEK	Menguji proses pengambilan dan pembukaan kunci dekripsi (*Data Encryption Key*) setelah *login* berhasil.	Klien menerima encrypted_DEK dari server, kemudian mendekripsinya menggunakan kunci turunan. DEK (*plaintext*) terbukti hanya tersimpan di sessionStorage lokal peramban.	Valid
Enkripsi Sisi Klien (Create Vault)	Pengguna memasukkan data rahasia baru (misal: kredensial Facebook) dan menyimpannya ke dalam *vault*.	Data diproses menggunakan algoritma XChaCha20 secara lokal. Sistem terbukti hanya mengirimkan *ciphertext*, *nonce*, dan *salt* ke *database*.	Valid
Dekripsi Sisi Klien (Read Vault)	Pengguna membuka salah satu item dari daftar *vault* untuk melihat kredensial.	Modul Rust WASM mengambil DEK dari memori lokal untuk mendekripsi *ciphertext*. Data *plaintext* hanya muncul di layar pengguna dan tidak pernah dikembalikan atau diproses ulang oleh server.	Valid

Logout	Menguji keamanan *memory management* ketika pengguna menekan tombol *logout* atau sesi *browser* berakhir.	Sistem secara otomatis menghapus DEK dan seluruh materi kriptografi sensitive dari sessionStorage. Vault terkunci dan memerlukan password Kembali untuk akses.	Valid

VII. SIMPULAN

Berdasarkan hasil perancangan dan pengujian, penelitian ini berhasil mengimplementasikan aplikasi pengelola kata sandi berbasis *Zero-Knowledge Architecture* menggunakan modul Rust WebAssembly (WASM) yang memastikan seluruh komputasi kriptografi dieksekusi secara penuh di sisi klien. Integrasi algoritma Argon2id untuk derivasi kunci yang tahan terhadap serangan *brute-force* GPU serta XChaCha20-Poly1305 untuk enkripsi data simetris terbukti mampu memberikan perlindungan berlapis, di mana server murni hanya menerima dan menyimpan *ciphertext*, *nonce*, dan *salt*. Selain itu, pembuktian teknis pada inspeksi jaringan (*network payload*), basis data, dan penyimpanan sesi (*sessionStorage*) mengonfirmasi bahwa *master password* asli tidak pernah terekspos ke server, serta manajemen *Data Encryption Key* (DEK) yang bersifat *volatile* secara efektif berhasil memitigasi risiko eksploitasi memori, menjadikan sistem ini sebagai solusi manajemen kredensial yang tangguh, aman, dan teruji keandalannya.

UCAPAN TERIMA KASIH

Penulis menyampaikan terima kasih kepada semua pihak yang telah memberikan dukungan, bimbingan, dan bantuan selama proses penyelesaian penelitian ini, baik dalam aspek akademik maupun nonakademik. Kontribusi berupa arahan ilmiah, saran yang membangun, serta penyediaan fasilitas pendukung sangat membantu penulis dalam menyelesaikan penelitian hingga tahap akhir. Selain itu, dukungan moral dan motivasi yang diberikan selama pelaksanaan penelitian juga menjadi faktor penting dalam menjaga kelancaran serta kualitas penyusunan karya ini.Penulis berharap hasil penelitian ini dapat memberikan manfaat serta menjadi referensi bagi pengembangan penelitian pada masa yang akan datang.

REFERENSI

[1] R. A. Oliveira and H. M. N. da S. Oliveira, “From RockYou to RockYou2024: Analyzing Password Patterns Across Generations, Their Use in Industrial Systems and Vulnerability to Password Guessing Attacks,” *Journal of Internet Services and Applications*, vol. 16, no. 1, pp. 43–57, 2025, doi: 10.5753/jisa.2025.5041.

[2] W. Y. Aditama, I. R. Hikmah, and D. F. Priambodo, “Analisis Komparatif Keamanan Aplikasi Pengelola Kata Sandi Berbayar Lastpass, 1Password, dan Keeper Berdasarkan ISO/IEC 25010,” *Jurnal Teknologi Informasi dan Ilmu Komputer*, vol. 10, no. 4, p. 857, 2023, doi: 10.25126/jtiik.20231036544.

[3] E. Chatzoglou, V. Kampourakis, Z. Tsiatsikas, G. Karopoulos, and G. Kambourakis, “Keep Your Memory Dump Shut: Unveiling Data Leaks in Password Managers,” in *IFIP Advances in Information and Communication Technology*, 2024, pp. 61–75. doi: 10.1007/978-3-031-65175-5_5.

[4] H. Setiawan, R. Dijaya, J. Mojopahit, and B. Sidoarjo, *Buku Ajar Audit Sistem Informasi di Era Digital: Teori, Praktik, dan Tren Terkini Diterbitkan oleh UMSIDA PRESS*. 2024.

[5] V. Navalino, A. F. Wadjdi, Y. Asnar, R. Agus, and G. Gultom, “Securing the Internet of Battlefield Things with ChaCha20- Poly1305 Encryption Architecture for Resource-Constrained Devices,” vol. 42, no. 2, pp. 547–555, 2024.

[6] A. Cherry, “A Secure Password Manager Governance Framework for Web User Authentication,” no. April, 2024.

[7] M. Shirvanian, C. R. Price, M. Jubur, N. Saxena, S. Jarecki, and H. Krawczyk, “A hidden-password online password manager,” *Proceedings of the ACM Symposium on Applied Computing*, vol. 20, pp. 1683–1686, 2021, doi: 10.1145/3412841.3442131.

[8] H. Padalia, H. Patel, A. Deshmukh, M. Patil, A. Kumar, and N. Kumar Nrip, “A Study on Password Manager: Users’ Perspective,” in *2023 International Conference on Computational Intelligence for Information, Security*

and Communication Applications (CIISCA), Bengaluru, India: IEEE, Jun. 2023, pp. 72–75. doi: 10.1109/CIISCA59740.2023.00024.

[9] A. Daftardar, B. Reagen, and S. Garg, "SZKP: A Scalable Accelerator Architecture for Zero-Knowledge Proofs," in *Proceedings of the 2024 International Conference on Parallel Architectures and Compilation Techniques*, Long Beach CA USA: ACM, Oct. 2024, pp. 271–283. doi: 10.1145/3656019.3676898.

[10] Haroon Rashid Hammood Al Dallal and Wijdan Noaman Marzoog Al Mukhtar, "A QR Code Used for Personal Information Based On Multi-Layer Encryption System," *Int. J. Interact. Mob. Technol.*, vol. 17, no. 09, pp. 44–56, May 2023, doi: 10.3991/ijim.v17i09.38777.

[11] A. Sharma, S. Sharma, S. R. Tanksalkar, S. Torres-Arias, and A. Machiry, "Rust for Embedded Systems: Current State and Open Problems," in *Proceedings of the 2024 on ACM SIGSAC Conference on Computer and Communications Security*, Salt Lake City UT USA: ACM, Dec. 2024, pp. 2296–2310. doi: 10.1145/3658644.3690275.

[12] P. P. Ray, "An Overview of WebAssembly for IoT: Background, Tools, State-of-the-Art, Challenges, and Future Directions," *Future Internet*, vol. 15, no. 8, p. 275, Aug. 2023, doi: 10.3390/fi15080275.

[13] S. Islam, "JavaScript alternative (TypeScript) and its effectiveness in web development".

[14] Y. Duan, D. Wang, and Y. Fu, "Security Analysis of Master-Password-Protected Password Management Protocols".

[15] K. Divya and P. S. Uma Priyadarsini, "Securing IoMT data with Algorand blockchain, XChaCha20-Poly1305 encryption, and decentralized storage alternatives," *Scientific Reports*, vol. 15, no. 1, pp. 1–20, 2025, doi: 10.1038/s41598-025-08527-9.

[16] A. A. S. AlQahtani, "Key Derivation: A Dynamic PBKDF2 Model for Modern Cryptographic Systems," *Cryptography*, vol. 9, no. 2, 2025, doi: 10.3390/cryptography9020039.

[17] S. Eum, H. Kim, M. Song, and H. Seo, "Optimized Implementation of Argon2 Utilizing the Graphics Processing Unit," *Applied Sciences (Switzerland)*, vol. 13, no. 16, 2023, doi: 10.3390/app13169295.

[18] T. O. Oladoyinbo, "The Importance Of Data Encryption Algorithm In Data Security," vol. 11, no. 2, pp. 10–16, 2024, doi: 10.9790/0050-11021016.

[19] M. Golinelli, F. Bonomi, and B. Crispo, "The Nonce-nce of Web Security: An Investigation of CSP Nonces Reuse," *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, vol. 14399 LNCS, pp. 459–475, 2024, doi: 10.1007/978-3-031-54129-2_27.

[20] V. Björkén and T. Gustafsson, "Evaluating the Effectiveness of LSTM-Based Password Generation versus Traditional Password Cracking Techniques on RockYou: Reassessing an Old Dataset with New Tools," vol. TRITA – EE, no. 2025:0000, 2025.

[21] A. Atadoga, O. A. Farayola, and B. S. Ayinla, "a Comparative Review of Data Encryption," vol. 5, no. 2, pp. 447–460, 2024, doi: 10.51594/csitrj.v5i2.807.

[22] A. Maspupah, "Literature Review: Advantages and Disadvantages of Black Box and White Box Testing Methods," *Jurnal Techno Nusa Mandiri*, vol. 21, no. 2, pp. 151–162, 2024, doi: 10.33480/techno.v21i2.5776.

[23] A. S. Lubis and M. P. A. Ginting, "Pengujian Aplikasi Berbasis Web Data SKA Menggunakan Metode Black Box Testing," *Cosmic Jurnal Teknik*, vol. 1, no. 1, pp. 41–48, 2024.

Conflict of Interest Statement:
The author declares that the research was conducted in the absence of any commercial or financial relationships that could be construed as a potential conflict of interest.