ZK Age Verification Project Documentation

1. Executive Summary

- **Brief overview:** This project implements a Zero-Knowledge (ZK) age verification system using zk-SNARKs (specifically Groth16) and Circom. It demonstrates how a user can prove they meet an age requirement (e.g., 16+) to a service provider without revealing their actual age, enhancing user privacy.
- **Key objectives:**
- Implement a privacy-preserving age verification flow.
- Simulate a government entity issuing signed age credentials.
- Develop a local proof generation mechanism.
- Create a service provider that verifies ZK proofs.
- Integrate the flow using a Chrome extension simulation.
- **Expected outcomes:** A functional local demonstration showcasing the end-to-end ZK age verification process, allowing users to access age-restricted content privately.
- **Summary of methodology:** The system uses zk-SNARKs (Groth16) with circuits written in Circom. A simulated government backend issues Schnorr-signed credentials. A local proof server generates proofs using `snarkjs`. A service provider verifies these proofs against the government's public key and the circuit's verification key. A Chrome extension orchestrates the user-side interactions.
- **Estimated timeline:**
- **Week 1:** Literature review, problem searching, review of existing implementations, deciding on the architecture.
- **Week 2:** Mock implementation without credential verification.
- **Week 3:** Added credential verification using Schnorr signatures.
- **Week 4:** Full implementation combining different contributions.

2. Introduction

- **Background and context:** Traditional online age verification methods often compromise user privacy by requiring the disclosure of sensitive personal information like date of birth or identity documents. Zero-knowledge proofs (ZKPs) offer a cryptographic solution, allowing individuals to prove statements (like meeting an age threshold) without revealing the underlying data.
- **Problem statement:** Accessing age-restricted online content or services necessitates age verification, but current methods create significant privacy risks and potential for data misuse or tracking [10, 11]. There is a need for verification mechanisms that protect user privacy while fulfilling legal or service requirements.
- **Relevance and significance:** This project demonstrates a practical application of ZKPs to address the critical challenge of online privacy in age verification. By leveraging zk-SNARKs, it offers a blueprint for systems that allow users to maintain control over their personal data while interacting securely online [16].
- **Research questions:** How can zk-SNARKs and Schnorr signatures be effectively combined to create a secure and private age verification system? What are the practical considerations for implementing such a system involving credential issuance, local proof generation, and verification?

3. Objectives and Scope

- **Primary objective:** To design, implement, and demonstrate a functional Zero-Knowledge age verification system using zk-SNARKs (Groth16), Circom, and Schnorr signatures.
- **Secondary objectives:**
- Simulate a trusted government backend for issuing signed age credentials.
- Implement a local proof generation server capable of creating ZK proofs based on user credentials and circuit definitions.
- Develop a service provider backend capable of verifying submitted ZK proofs.
- Simulate user interaction via a Chrome browser extension.

- **Scope:** The project focuses on a local demonstration environment. It includes the core components: credential issuance, proof generation, and proof verification. The circuits are designed specifically for age verification using Schnorr signatures.
- **Limitations and assumptions:**
- The system operates in a simulated local environment (`localhost`).
- The "government" is a mock backend, and user authentication is simplified.
- Credential storage and management in the extension are basic.
- The proof server runs locally, simulating generation within a secure user environment (like the extension itself in a production scenario).
- The project uses the Groth16 proving system, which requires a trusted setup (assumed to be performed correctly for the provided `.zkey` files).
- Error handling and security hardening are basic, suitable for a demonstration.

4. Literature Review

Age Verification Using Zero-Knowledge Proofs: A Literature Review

Zero-knowledge proofs have emerged as a powerful cryptographic technique to enhance privacy while maintaining trust in digital identity verification. This literature review examines the development, implementation, and challenges of age verification systems using zero-knowledge proofs, with a particular focus on zk-SNARKs (Zero-Knowledge Succinct Non-Interactive Arguments of Knowledge).

Background and Fundamentals

Zero-knowledge proofs (ZKPs) allow a prover to convince a verifier that a statement is true without revealing any additional information beyond the validity of the statement itself. In the context of age verification, ZKPs enable users to prove they meet age requirements (e.g., "I am over 18") without disclosing their actual age or date of birth[16]. This technology addresses a critical privacy challenge in online environments where age verification is required but excessive personal data collection poses risks.

The Need for Private Age Verification

Traditional age verification methods often require users to share sensitive identity documents or personally identifiable information. As Rosenberg et al. note, countries such as Australia, the EU, and the UK require identification via credit cards or official IDs to access age-restricted content, creating significant privacy and tracking risks[10]. The Digital Personal Data Protection Act in India similarly imposes obligations that could force businesses to verify the age of every person they interact with, potentially breaking "the internet as we know it"[11].

Technical Approaches to Zero-Knowledge Age Verification

Hash Chain Verification

One of the simpler approaches to age verification uses hash chains, as demonstrated by Sterjev (2023). In this method, a user generates a hash chain starting from a seed value, with the number of hash operations corresponding to their actual age. To prove they are above a certain age threshold (e.g., 18), they provide an intermediate hash value that can be further hashed the remaining number of times to match their committed age value[1].

```
const actual_age = 49;

const age_to_prove = 18;

const proof = hash(seed, actual_age - age_to_prove);
```

```
const hashed_age = hash(seed, actual_age);
```

The verifier can then hash the proof value exactly `age_to_prove` times to check if it matches the committed age value, without learning the actual age[1].

zk-SNARKs for Age Verification

More sophisticated approaches use zk-SNARKs, which provide greater flexibility and security. Sterjev demonstrates implementing age verification using libraries like snarkjs and circom, where a circuit validates that a user's actual age exceeds the required threshold[1]. This approach allows for non-interactive proofs that can be verified efficiently without further interaction with the prover.

The circuit for age verification can be designed as simple as:

```
template AgeVerifier() {

signal private input actual_age;

signal input age_to_prove;

signal output out;



out = age_to_prove ? 1 : 0;

}
```

The verification process then checks the cryptographic proof without learning the actual age value[1][9].

Range Proofs

Zero-knowledge range proofs (ZKRPs) are particularly well-suited for age verification as they allow proving that a value lies within a specific interval. As detailed in "SoK: Zero-Knowledge Range Proofs," these proofs are essential in applications where a user needs to demonstrate that a secret value (such as age) falls within certain bounds without revealing the exact value[3].

Bulletproofs, described by Bünz et al., offer "short proofs for confidential transactions" and are "especially well suited for efficient range proofs on committed values"[12]. They enable proving that a committed value is in a range using logarithmic proof size, making them practical for implementation in resource-constrained environments[19].

Implementation Frameworks and Libraries

Circom and snarkjs

The Circom domain-specific language and snarkjs JavaScript library have become popular tools for implementing zk-SNARKs, including age verification systems. Sterjev provides a detailed implementation using these tools, showing how to compile circuits, generate proving keys, and verify proofs[1]. This framework supports the Groth16 proving scheme, which is efficient but requires a trusted setup[9].

ZoKrates

ZoKrates provides another approach to implementing age verification. A Stack Overflow question details an implementation where a user can prove their age is over 21 without revealing their date of birth[8]. The implementation uses a simple circuit that checks if the birth year and century satisfy the age requirement:

```
def main(pubName, private yearOfBirth, private centuryOfBirth):

x = if centuryOfBirth == 19 then 1 else 0 fi

y = if yearOfBirth < 98 then 1 else 0 fi

// Additional checks

return result
```

This implementation also addresses the challenge of preventing proof reuse through nullifiers-hashes that can be registered to prevent multiple uses of the same proof[8].

Real-World Systems and Applications

Yoti Keys

Yoti Keys represents a practical application of age verification that prioritizes privacy. The system allows users to "verify their age once and gain continued access to an ecosystem of websites without having to prove their age again," even when using private browsing modes[4]. While not explicitly using zk-SNARKs, Yoti's approach demonstrates the growing market for privacy-preserving age verification solutions.

zk-creds

Rosenberg et al. present "zk-creds," a protocol that uses general-purpose zero-knowledge proofs to convert existing identity documents into anonymous credentials without requiring coordination with issuing authorities. Their system demonstrates anonymous access to age-restricted videos in under 150ms using passport data, showing the practicality of these approaches[10].

Challenges and Considerations

Trust and Credential Issuance

A significant challenge with zero-knowledge age verification is establishing trust in the underlying credentials. As noted in a LinkedIn post by Vishwas Anand Bhushan, "simply verifying the ZK proof isn't enough!" The verifier must also confirm that the proof was generated using credentials from a trusted issuer[5]. This highlights the need for comprehensive circuit design that verifies issuer signatures, credential validity, and user authentication alongside the age verification itself.

Proving Without Revealing

Implementing age verification without revealing any additional information presents unique cryptographic challenges. On Crypto Stack Exchange, users discuss approaches like using hash chains or elliptic curve cryptography with Pedersen Commitments to prove age requirements without revealing birth dates[20]. These discussions highlight the ongoing exploration of different mathematical approaches to the same fundamental problem.

Performance and Efficiency

Performance remains a key consideration for practical implementation. A comparative study of zkSNARK libraries for blockchains found that even for simple age verification circuits, there are significant differences in storage

requirements and performance between implementations. For instance, Circom required 4KB for circuit files compared to 139KB for ZoKrates, though both had similar proof sizes[9].

Recent Developments and Future Directions

Anonymous Credentials with Enhanced Features

Recent work has focused on extending anonymous credential systems with additional features. The "Revocable TACO" system introduces threshold-based anonymous credentials with opening and revocation capabilities, allowing for accountability in systems that provide anonymity[18]. This addresses scenarios where a user might default on obligations after anonymously proving age or income requirements for services like loans.

Attribute Encoding Optimizations

Camenisch and Groß propose encoding binary and finite-set attributes as prime numbers, using the divisibility property for efficient proofs of their presence or absence[7]. This approach significantly improves the efficiency of selective disclosure in credential systems, making them more suitable for applications like electronic identity cards that require verification of multiple attributes including age.

Conclusion (from Literature Review)

Zero-knowledge proofs, particularly zk-SNARKs, offer promising solutions for private age verification in digital environments. The literature shows significant progress in both theoretical approaches and practical implementations, with systems ranging from simple hash-chain verification to sophisticated anonymous credential frameworks.

Key challenges remain in balancing privacy with trust, ensuring efficient verification on resource-constrained devices, and establishing proper governance for credential issuance. However, the continued advancement of libraries, encoding techniques, and proof systems suggests that zero-knowledge age verification will become increasingly practical and widespread.

As one article aptly states, zero-knowledge proofs allow users to keep their "treasure chest of personal information locked away" while still proving necessary attributes like age[16], offering a path toward digital interactions that respect both security requirements and user privacy.

5. Resource Used

- **Programming Language:** JavaScript (Node.js for backends, browser JS for extension)
- **Frameworks/Libraries:**
- Node.js
- Express.js (for backend servers)
- `snarkjs` : ZK-SNARK implementation library (Groth16 proving and verification).
- `circomlib` / `circomlibjs` : Cryptographic primitives for Circom circuits (used implicitly via `mimc_utils.js` for Schnorr).
- `jsonwebtoken` : For signing/verifying credentials during transport.
- `cors` : For enabling cross-origin requests between local servers.
- `axios` : For making HTTP requests between servers (Service Provider to Government).
- `crypto` (Node.js built-in): For cryptographic operations.
- **ZK Tools:**
- Circom: Domain-specific language for writing arithmetic circuits.
- Groth16: zk-SNARK proving system used.
- **Development Environment:** Visual Studio Code, Node.js runtime, Chrome browser.
- **Hardware:** Standard development machine capable of running Node.js servers and Chrome.

6. Methodology

- **Research design/experimental setup:** The project uses a simulation-based approach with multiple locally running servers mimicking real-world entities.
- **Government Backend (`localhost:3001`):** Simulates a trusted authority. Issues JWT-wrapped credentials containing the user's age and a Schnorr signature (`{ signature, nonce, message: age }`) generated using its private key. Exposes its public key via an API.
- **Circuit Server (`localhost:3002`):** (Development only) A static server providing circuit files (`.wasm` , `.zkey` , verification key) to the proof generator. In production, these would be bundled.
- **Proof Server (`localhost:3003`):** Runs locally on the user's machine (simulating secure local execution). Receives the credential, extracts necessary inputs (age, signature details, public key), and uses `snarkjs.groth16.fullProve` with the circuit's WASM and ZKey files to generate the ZK proof and public signals.
- **Service Provider (`localhost:3000`):** Represents the website requiring age verification. It fetches the government's public key. When a user attempts access, it requests a ZK proof. Upon receiving the proof and public signals, it uses `snarkjs.groth16.verify` with the verification key. It checks the proof's validity and verifies that the public key in the public signals matches the fetched government public key.
- **Chrome Extension:** Simulates the user agent. It interacts with the Government Backend to get credentials, stores them (using `chrome.storage`), interacts with the Service Provider to initiate verification, sends the credential to the local Proof Server, receives the proof, and submits it to the Service Provider.
- **Tools, technologies, and frameworks:** As listed in Section 5. The core is the combination of Circom for circuit definition (`schnorr_age_verification.circom`), `snarkjs` for proof generation/verification, and Node.js/Express for the server infrastructure. Schnorr signatures are used for credential signing, compatible with the circuit.
- **End-to-End Flow:**

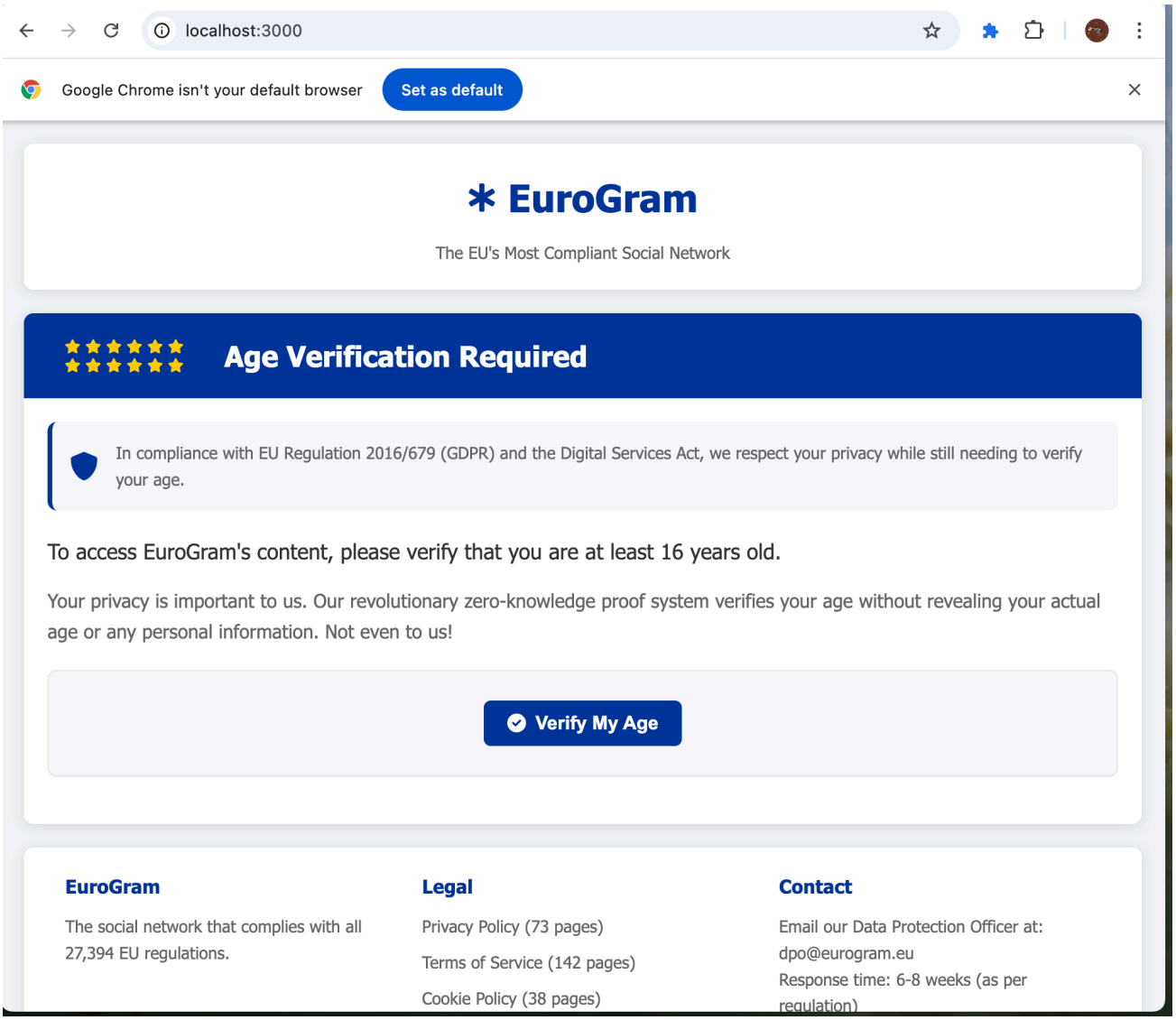
1. User visits the Government Backend website (`localhost:3001`) and authenticates (mocked).
2. Government Backend issues a signed credential (JWT containing age, Schnorr signature, public key).
3. Chrome extension stores the credential.
4. User visits the Service Provider website (`localhost:3000`).
5. Service Provider requires age verification and communicates the requirement (e.g., fixed age 16+) to the extension.
6. Extension sends the stored credential to the local Proof Server (`localhost:3003`).
7. Proof Server generates the ZK proof using the credential data and circuit files. The public signals include `isVerified` (1 if age >= requirement) and the `governmentPublicKey` .
8. Proof Server returns the `proof` and `publicSignals` to the extension.
9. Extension submits the `proof` and `publicSignals` to the Service Provider's `/api/verify-proof` endpoint.
10. Service Provider verifies the proof using the verification key and checks if the `governmentPublicKey` in the public signals matches its known key. If valid and `isVerified` is 1, access is granted.

- **Development vs. Production Considerations:**
- **Circuit Distribution:** Development uses a central `circuit-server` . Production would bundle files with the extension or use decentralized storage (e.g., IPFS) with local caching.
- **Proof Generation:** Development uses a separate `proof-server` . Production would ideally perform proof generation directly within the secure context of the browser extension to avoid exposing credentials externally.
- **Trust Model:** Development relies on trusting local servers. Production relies primarily on trusting the initial government credential issuance; subsequent steps are trustless due to ZK proofs and local generation.
- **Data collection methods:** Not applicable (simulation-based).
- **Risk assessment and mitigation strategies:**
- **Credential Security:** Risk of credential theft from browser storage. Mitigation: Use secure storage (`chrome.storage`), potentially encrypt credentials.
- **Proof Replay:** Risk of reusing a valid proof. Mitigation: Implement nullifiers within the circuit and a registry on the Service Provider side (as discussed in literature [8], but not implemented in this demo).
- **Private Key Security:** Government private key compromise would break the system. Mitigation: Secure key management practices (HSMs in real-world).

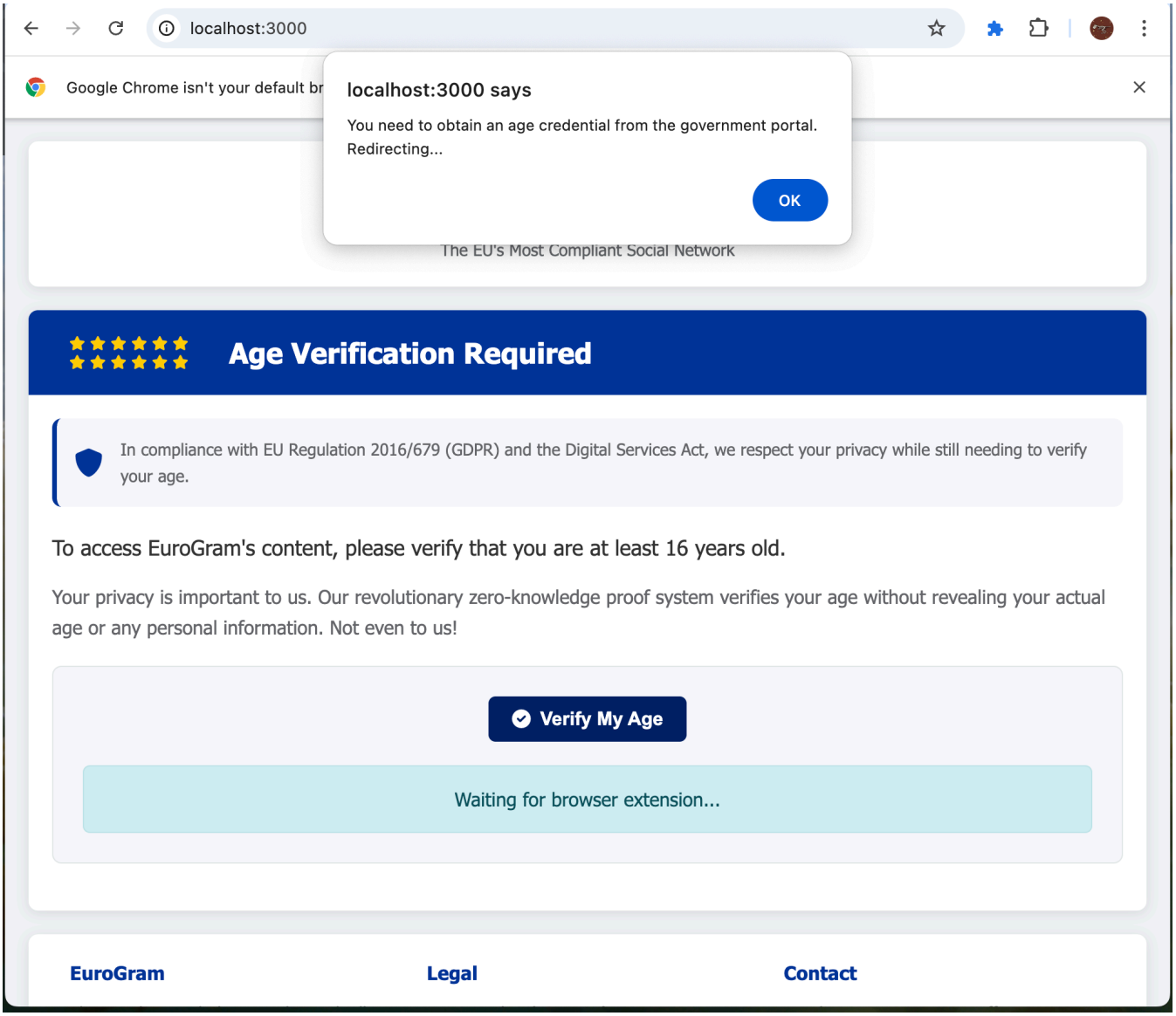
- **Circuit Integrity:** Malicious circuit could leak private inputs. Mitigation: Auditing circuits, using trusted setup ceremonies.

7. Results

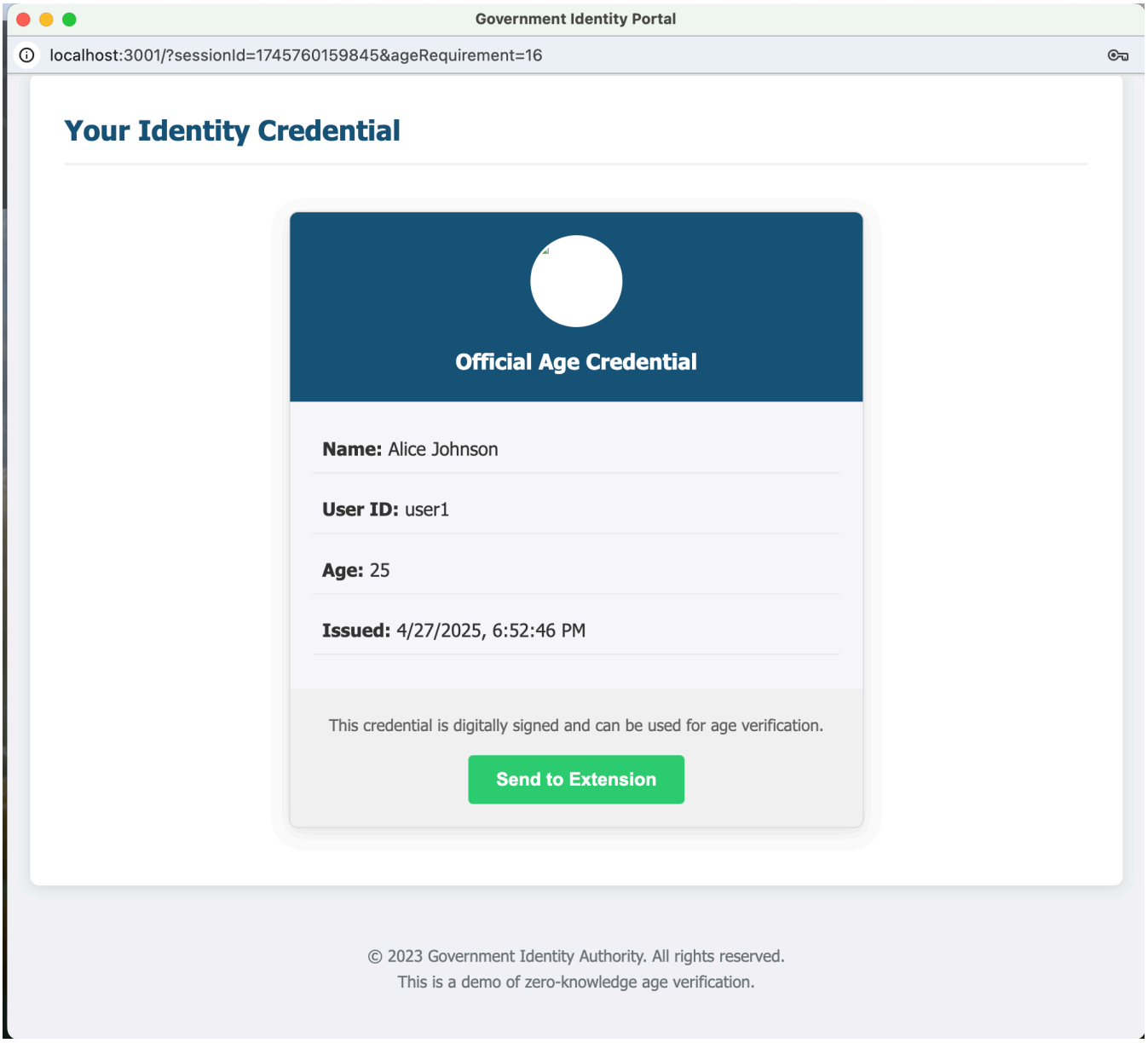
- **Summary of preliminary findings:** The implemented system successfully demonstrates the ZK age verification flow. Users can obtain a credential, generate a proof locally, and submit it to the service provider.
- **Interpretation of the processed data:** The service provider successfully verifies valid proofs generated using credentials from the mock government, confirming the user meets the age requirement (16+) without learning the actual age. The verification includes checking the SNARK proof itself and confirming the government public key embedded in the proof's public signals.
- **Screenshots/visual results:**
- User visits the Service provider :



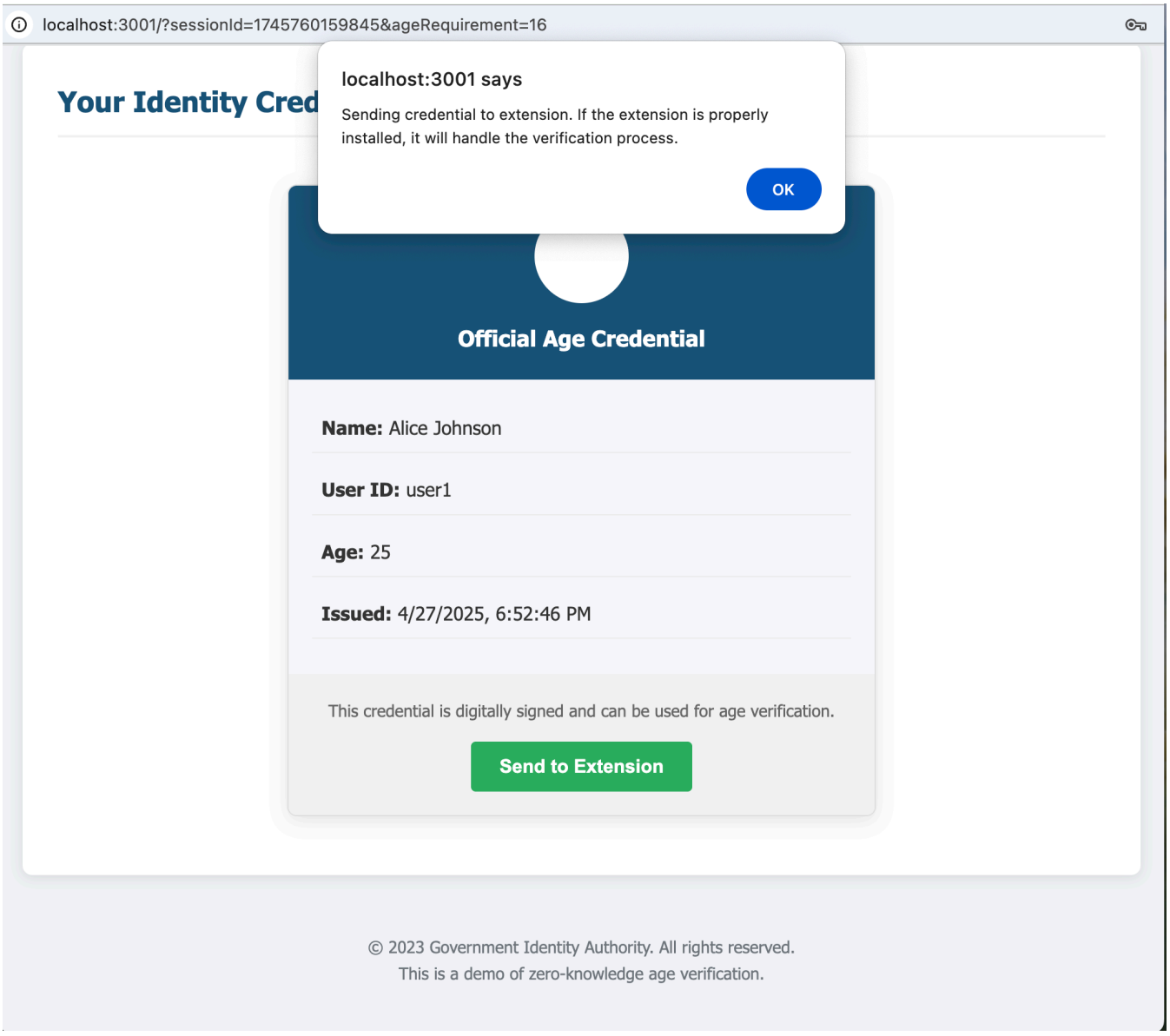
Extension Prompts the user :



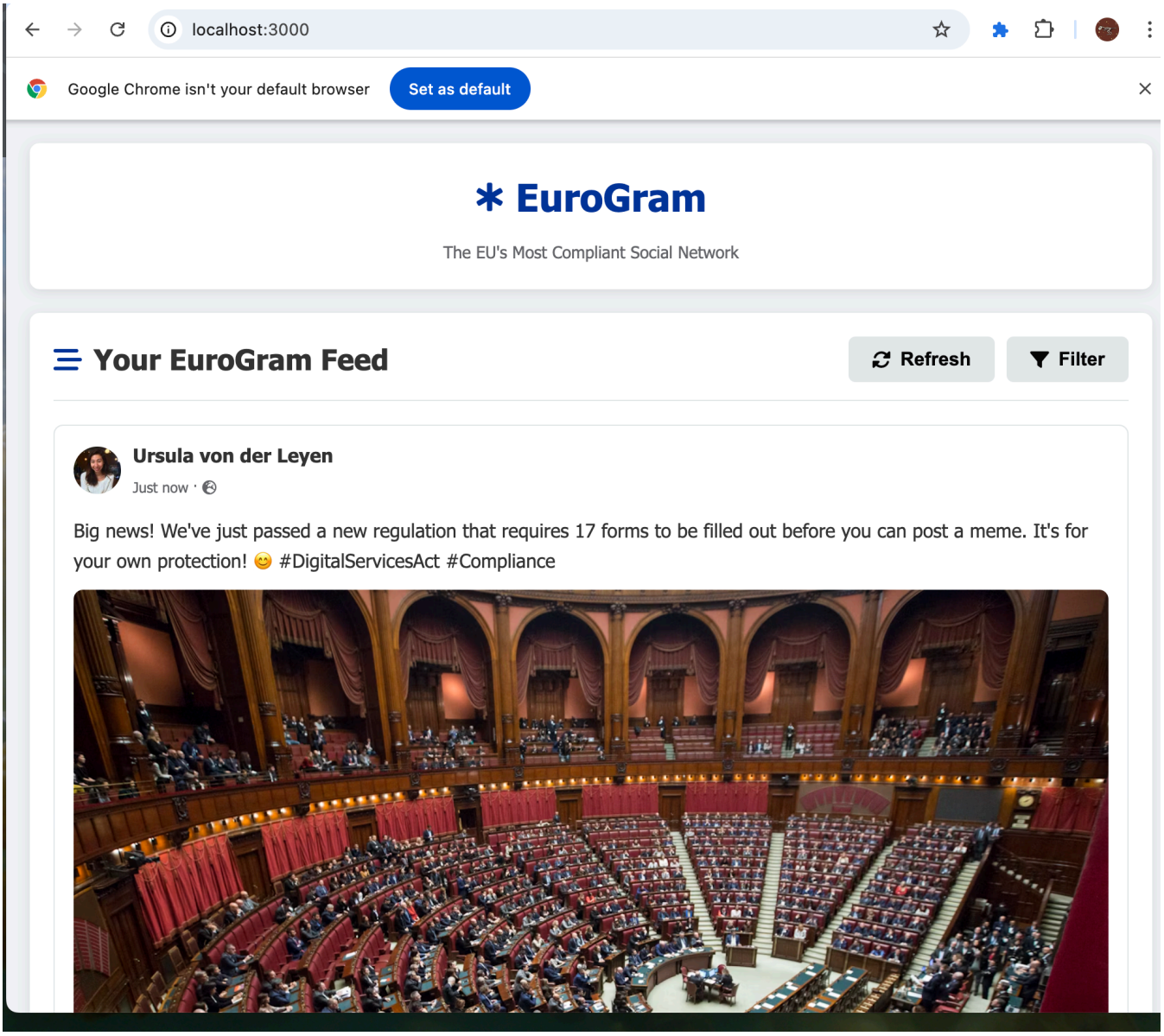
User visits the govt website and logins:



The credentials are sent to the extension:



After Proof Generation, the extension feeds the proof to the service provider:



8. Analysis

- **Discussion of observed trends, insights from results:** The system validates the feasibility of using zk-SNARKs (Groth16) and Schnorr signatures for privacy-preserving age verification in a web context. The separation of concerns (issuance, proof generation, verification) works effectively in the simulated environment. The local proof generation step is crucial for maintaining user privacy, as the raw credential (containing the actual age) does not leave the user's control after issuance.
- **Comparisons with existing approaches or baselines:** Compared to traditional methods (sharing ID scans, DOB), this ZK approach significantly enhances privacy. Unlike simple hash chains [1], zk-SNARKs allow verification against a public key signature, binding the age proof to a trusted issuer. The performance, while not rigorously benchmarked here, relies on `snarkjs` capabilities, which are generally suitable for web applications [9].
- **Limitations of the analysis and external factors considered:** The analysis is based on a local simulation. Real-world deployment would face challenges like performance on diverse user devices, secure key distribution, robust credential management, and the need for a widely accepted trust framework for credential issuers. The lack of nullifiers limits protection against proof replay in this specific demo. The reliance on Schnorr signatures requires compatible circuit libraries.

9. Future Work

- **Planned expansions or next steps:**

- **Implement Nullifiers:** Add unique nullifiers to the circuit and verification logic to prevent proof replay attacks [8].
- **Explore Other Proof Systems:** Investigate alternatives like Bulletproofs [12, 19] for potentially smaller proof sizes or different trust assumptions, or PLONK for universal trusted setups.
- **Dynamic Age Requirements:** Modify the circuit and flow to support variable age requirements set by the service provider, rather than a fixed one.
- **Explore ways to mitigate the timing based identity linking**
- **Credential Revocation:** Implement a mechanism for revoking compromised or invalid credentials [18, 26].
- **Benchmarking:** Conduct performance testing on various devices for proof generation and verification times.
- **UI/UX Improvements:** Enhance the Chrome extension interface for better usability.
- **New features, enhancements, or alternative strategies:** Consider using Decentralized Identifiers (DIDs) and Verifiable Credentials (VCs) standards alongside ZKPs for a more interoperable identity framework. Explore attribute encoding optimizations [7].
- **Timeline for future improvements:** Cant say, but will take atleast a week of full work to arrive at good solutions.

10. Conclusion

- **Summary of key points:** This project successfully implemented a Zero-Knowledge age verification system using zk-SNARKs (Groth16), Circom, and Schnorr signatures. It demonstrates a privacy-preserving flow involving credential issuance by a mock government, local proof generation, and verification by a service provider, orchestrated via a Chrome extension simulation.
- **Final remarks on project feasibility and significance:** The project confirms the technical feasibility of ZK-based age verification for web applications. While challenges remain for production deployment (performance, security, trust frameworks), the approach offers significant privacy advantages over traditional methods, aligning with the principles discussed in the literature [5, 16]. It serves as a valuable proof-of-concept for building more private and user-centric digital identity solutions.

11. References

[1] https://www.linkedin.com/pulse/age-zero-knowledge-proofs-marjan-sterjev

[2] https://www.nttdata.com/global/en/insights/focus/2024/what-is-zero-knowledge-proof

[3] https://drops.dagstuhl.de/storage/00lipics/lipics-vol316-aft2024/LIPIcs.AFT.2024.14/LIPIcs.AFT.2024.14.pdf

[4] https://www.yoti.com/blog/yoti-keys-private-seamless-anonymous-age-verification/

[5] https://www.linkedin.com/posts/vishwas-anand-bhushan-bab55576_zk-zkp-zk-activity-7261225902129221632-WhvA

[6] https://idemix.wordpress.com/scenarios/

[7] https://eprint.iacr.org/2010/496.pdf

[8] https://stackoverflow.com/questions/51605174/writing-a-circuit-in-zokrates-to-proof-age-is-over-21-years

[9] https://ceur-ws.org/Vol-3791/paper7.pdf

[10] https://eprint.iacr.org/2022/878.pdf

[11] https://exmachina.in/10/04/2024/age-tokens/

[12] https://eprint.iacr.org/2017/1066.pdf

[13] https://crypto.stackexchange.com/questions/68791/bulletproofs-range-proof-of-value-provided-from-trusted-third-party

[14] https://docs.sui.io/references/framework/sui/groth16

[15] https://www.linkedin.com/pulse/zero-knowledge-proof-developers-guide-example-gourav-patidar-3od3f

[16] https://www.galaxy.com/insights/perspectives/zero-knowledge-proofs-the-magic-key-to-identity-privacy/

[17] https://developers.id.me/documentation/identity-gateway/attribute-exchange/age

[18] https://dl.acm.org/doi/pdf/10.1145/3634737.3637656

[19] https://github.com/sdiehl/bulletproofs

[20] https://crypto.stackexchange.com/questions/96232/zkp-prove-that-18-while-hiding-age

[21] https://tokenminds.co/blog/knowledge-base/zk-snarks-in-practice

[22] https://metaschool.so/articles/zero-knowledge-proofs-understanding-basics/

[23] https://eprint.iacr.org/2024/1153.pdf

[24] https://eprint.iacr.org/2024/430.pdf

[25] https://ondato.com/blog/onage/

[26] https://identity.foundation/revocation/non-revocation-token/

[27] https://onlinelibrary.wiley.com/doi/full/10.1002/spy2.401

[28] https://veridas.com/en/what-is-zero-knowledge-proof/

[29] https://www.mystenlabs.com/blog/zero-knowledge-range-proofs

[30] https://hyperverge.co/blog/how-does-age-verification-works-online/

[31] https://eprint.iacr.org/2022/878.pdf (Duplicate of [10])

[32] https://www.di.ens.fr/~nitulesc/files/Survey-SNARKs.pdf

[33] https://hyperledger-fabric.readthedocs.io/en/latest/idemix.html

[34] https://www.zurich.ibm.com/pdf/csc/Identity_Mixer_Nov_2015.pdf

[35] https://github.com/IBM/idemix

[36] https://eprint.iacr.org/2012/298.pdf

[37] https://www.developer.tech.gov.sg/communities/events/stack-meetups/past-webinars/zero-knowledge-proof-part-2.html

[38] https://www.idology.com/use-cases/age-verification/

[39] https://www.scribd.com/document/551916912/2-medical-certificate-to-prove-age

[40] https://www.ieee-security.org/TC/SPW2025/ConPro/papers/liu-conpro25.pdf

[41] https://github.com/Zokrates/ZoKrates/issues/94

[42] https://experience.idemia.com/identity-proofing/

[43] https://fisher.wharton.upenn.edu/wp-content/uploads/2020/09/Thesis_Terrence-Jo.pdf

[44] https://arxiv.org/pdf/2310.15934.pdf

[45] https://www.mdpi.com/2078-2489/15/8/463

[46] https://info.cs.st-andrews.ac.uk/student-handbook/files/project-library/cs4796/gf45-Final_Report.pdf

[47] https://www.rjwave.org/ijedr/papers/IJEDR2101020.pdf

[48] https://eprint.iacr.org/2024/1348.pdf

[49] https://www.usenix.org/system/files/usenixsecurity23-hesse.pdf

[50] https://eprint.iacr.org/2025/172.pdf

[51] https://blog.pantherprotocol.io/bulletproofs-in-crypto-an-introduction-to-a-non-interactive-zk-proof/

[52] https://www.rareskills.io/post/bulletproofs-zk

[53] https://www.youtube.com/watch?v=DDe29vLNhz8

[54] https://tlu.tarilabs.com/cryptography/the-bulletproof-protocols.html

[55] https://www.rareskills.io/post/groth16

[56] https://crypto.stackexchange.com/questions/96232/zkp-prove-that-18-while-hiding-age (Duplicate of [20])

[57] https://cryptopapers.info/assets/pdf/bulletproofs.pdf

[58] https://github.com/matter-labs-archive/Groth16BatchVerifier

[59] https://eprint.iacr.org/2020/735.pdf

[60] https://docs.iota.org/developer/cryptography/on-chain/groth16

[61] https://www.law.cornell.edu/cfr/text/20/219.21

[62] https://www.yoti.com/blog/yoti-keys-private-seamless-anonymous-age-verification/ (Duplicate of [4])

[63] https://www.idenfy.com/glossary/age-verification-system/

[64] https://www.nobroker.in/forum/what-are-the-age-proof-documents/

[65] https://www.linkedin.com/pulse/age-zero-knowledge-proofs-marjan-sterjev (Duplicate of [1])

[66] https://americarenewing.com/issues/identity-on-the-internet-protecting-children-and-privacy-and-building-a-proof-of-humanity-regulatory-regime-for-an-ai-driven-internet/

[67] https://blog.stratumn.com/zero-knowledge-proof-of-age-using-hash-chains-d034d262bd5f

[68] https://crypto.stanford.edu/bulletproofs/

[69] https://www.zellic.io/blog/gnark-bug-groth16-commitments

[70] https://ceur-ws.org/Vol-3791/paper7.pdf (Duplicate of [9])

[71] https://doc-internal.dalek.rs/bulletproofs/notes/range_proof/index.html

12. Team Contributions

- **Chirag Aggarwal (2021EEB1161):** ZK Circuit design and overall system architecture.
- **Pranav Bali (2021EEB1193):** Literature Review, Chrome Extension Frontend and Backend integration.
- **Jugal Alan (2024AIM1004):** Mock implementation development, Proof Server implementation, Circuit Files management.