

# **Decentralized Identification with Press ID Badge**

**By: Patrick O. Ingle, July 16, 2025**

**Empowering Secure, Passwordless Authentication Through Local Cryptographic Signing**

## **Abstract**

This whitepaper presents the Press ID Badge browser extension, a decentralized identification tool designed to eliminate reliance on passwords and centralized credential stores. By leveraging local cryptographic signing—where private keys never leave the user’s device—Press ID Badge offers a secure, auditable, and user-respecting alternative to traditional login and verification systems. We explore its implementation, benefits, and implications for digital trust, especially in high-sensitivity environments such as journalism, decentralized governance, and credential-backed interactions.

## **Introduction**

Digital identity remains fraught with compromise, complexity, and centralization. The dominant user-password model breeds weak credentials, dependency on trust intermediaries, and vulnerability to phishing, leaks, and social engineering. Press ID Badge tackles this problem head-on, enabling:

- Service message signing via locally stored private keys
- Auth workflows based on verifiable credentials and DID documents
- Passwordless authentication fortified with cryptographic assurance
- Seamless integration into content-script-secured browsers via modular architecture



## Core Features & Safety Guarantees

The encryption used in the extension is: **RSASSA-PKCS1 with a SHA-256 hash**.

### 1. Local Private Key Signing

- Press ID Badge generates and stores the user's signing key **locally**.
- All cryptographic operations—RSA, ECDSA, etc.—happen **in-browser**, using Web Crypto or WASM-backed modules.
- No external transmission or exposure of the private key.

### 2. Signed Service Messages

- Requests, attestations, and credential presentations are **digitally signed**.
- Services verify these signatures using publicly shared keys or DID documents.
- Reduces attack surface for MITM and replay attacks.

### 3. Passwordless Logins via Credential Exchange

- Users authenticate by presenting signed, verifiable credentials.
- No passwords to forget, reuse, or reset.
- Compatible with W3C Verifiable Credentials and DID Auth flows.

### 4. 2FA Without the Fuss

- Built-in support for hardware key challenge signing (e.g., WebAuthn fallback).
- Optional biometric prompts via browser APIs.



## Decentralized Architecture Overview

```
graph TD;
    User["User (Browser)"]
    DID["Decentralized Identifier (DID)"]
    VC["Verifiable Credential"]
    Service["Service (Verifier)"]

    User -->|Signs Message| Service
    Service -->|Requests Proof| User
    User -->|Presents VC| Service
    User --> DID
    Service --> DID
```

- **No centralized authority** governs credential issuance or user identity.
- Credentials are **portable**, **revocable**, and **selectively disclosable**.

## Technical Design

- Modular architecture with separation of **content scripts**, **background logic**, and **signature engine**
- CSP-compliant script injection workflows
- Enforced origin trust boundaries for messaging and credential presentation
- JSON-based JWK support with RSA/ECDSA encoding utilities
- Compatibility across Chrome, Brave, and Firefox

## Security Benefits

Feature	Traditional Auth Model	Press ID Badge
Password Resets	Frequent, brittle	Eliminated
Phishing Risk	High	Cryptographically mitigated
Credential Storage	Server-side, breach-prone	Local-only, user-owned
Key Rotation	Manual, error-prone	Built-in with DID updates
2FA Bypass	Possible via phishing	Challenge-response signed locally

## Use Cases

- Newsroom access control via Press ID Badge verification
- Independent journalists presenting verified credentials to secure interviews or sensitive locations
- DAO members authenticating cryptographically without revealing personal data
- Enterprise users reducing IT overhead by using decentralized IDs for internal tooling



## Future Directions

- zk-proof-backed disclosures for pseudonymous participation
- Integration with reputation systems and selective transparency layers
- Open-source tooling for cross-language validators (PHP, Node.js, etc.)



## Conclusion

Press ID Badge represents a paradigm shift in how we verify identity and trust online. By empowering users with tools to sign and authenticate locally, the extension moves us closer to a world where credentials are **secure**, **self-sovereign**, and **independent of centralized control**.

---

## Self-Declaration of Cryptographic Functionality

**App Name:** Press-ID-Badge

**Developer:** Amerione Corporation

**Functionality:** Digital Signature Generation and Verification

**Date:** August 4, 2025

**Prepared by:** Patrick [Systems Architect, Compliance Lead]

## Cryptographic Use Summary

- **Algorithms Used:**
  - Signature Scheme: RSASSA-PKCS1-v1\_5
  - Hash Function: SHA-256
- **Purpose:**
  - Signing badge payloads using a user-selected RSA private key
  - Verifying signatures using the corresponding RSA public key
- **Key Characteristics:**
  - Format: PKCS #8 (private), SPKI (public)
  - Size: 2048 bits
  - Selection: User-provided or generated via Web Crypto API
  - Storage: Keys are not persisted; used in-memory only
- **Implementation:**
  - Web Crypto API (`crypto.subtle.sign`, `crypto.subtle.verify`)
  - No custom cryptographic primitives
  - No encryption or decryption functionality





## Export Classification Statement

This application uses **standard cryptographic algorithms** solely for **authentication and integrity verification**. It does not:

- Implement proprietary or non-standard cryptography
- Facilitate encrypted communications or data storage
- Include military-grade or surveillance functionality

Under **U.S. Export Administration Regulations (EAR)**, this use case qualifies for **Category 5 Part 2 exemption** as defined by BIS.

## Declaration for App Store Connect

- **Does the app use encryption?**  Yes
- **Is the encryption limited to standard authentication or signature verification?**  Yes
- **Is the app exempt from ERN filing?**  Yes, under Category 5 Part 2
- **Is the encryption functionality built into the OS or standard APIs?**  Yes (Web Crypto API)

## Supporting Documentation (Optional Uploads)

- Screenshot of `crypto.subtle.sign()` and `crypto.subtle.verify()` usage

```
const signature = await crypto.subtle.sign(  
  {  
    name: "RSASSA-PKCS1-v1_5",  
    hash: { name: "SHA-256" }  
  },  
  key,  
  encodedMsg  
);
```

```
const isValid = await crypto.subtle.verify(  
  { name: "RSASSA-PKCS1-v1_5" },  
  publicKey,  
  signatureBytes,  
  messageBytes  
);
```

- Source code excerpt showing RSASSA-PKCS1-v1\_5 with SHA-256 (source code at <https://github.com/verifiedpress/press-id-badge>)

```
async function signMessage(privateKeyPem, message) {  
  
  const key = await importRsaPrivateKey(privateKeyPem);  
  
  const encodedMsg = new TextEncoder().encode(message);  
  
  const signature = await crypto.subtle.sign( {  
  
    name: "RSASSA-PKCS1-v1_5",    hash: { name: "SHA-256" } },  
  
    key, encodedMsg );  
  
  const hexSignature = Array.from(new Uint8Array(signature)) .map(b => b.toString(16).padStart(2, "0"))  
    .join("");  
  
  return hexSignature;  
  
}
```