Supply Chain Identity Verification through Winter Auth: A Technical Guide

Supply chain integrity depends on a fundamental question, “**who verified this information?**”

When a field supervisor confirms that organic farming practices were followed, or when a quality inspector approves a harvest for export, the value of that verification is only as strong as the identity behind it. Traditional paper-based systems can't answer this question reliably, but blockchain-based identity verification using Winter Auth creates an immutable record of exactly who approved each supply chain event.

This technical guide walks through implementing a complete supply chain identity verification system using Winter Auth, ADA Handles for **identity anchoring**, and Winter Protocol for immutable record creation. The implementation covers everything from wallet connection to biometric verification, creating a secure foundation for agricultural traceability that regulatory bodies can trust.

System Architecture Overview

The implementation creates a three-tier system: administrative management, identity verification, and supply chain recording. Company administrators manage a main Handle.me NFT and mint sub-handles for field supervisors. Each sub-handle becomes linked to a specific employee through biometric verification. When supervisors need to record supply chain events, they must authenticate using Winter Auth before submitting data through Winter Protocol.

This architecture ensures that every supply chain record includes cryptographic proof of who created it. Regulatory auditors can verify not just what happened, but exactly who verified that it happened. The Handle.me sub-handle system provides human-readable identity management while maintaining the security properties needed for regulatory compliance, in system, only verified credentials are enabled to submit blockchain records.

The modular design allows companies to implement the system gradually. Organizations can start with critical checkpoints, harvest certification, quality inspection, or export approval, and expand coverage as they gain experience with the technology. Each component integrates with existing business processes while adding the verification capabilities that modern supply chains require.

Wallet Connection and Handle.me Integration

The foundation of the system starts with cardano wallet connectivity using MeshSDK and the CIP-30 standard. CIP-30 defines the web-based communication bridge that allows applications to interface with Cardano wallets through standardized JavaScript APIs. This ensures compatibility across different wallet implementations while providing secure transaction signing capabilities.

TypeScript

```
import { BrowserWallet } from '@meshsdk/core';

// Initialize wallet connection
const connectWallet = async () => {
  try {
    // Connect to user's preferred CIP-30 compatible wallet
    const wallet = await BrowserWallet.enable('eternl'); // or
nami, yoroi, etc.

    // Get wallet address for Handle.me verification
    const addresses = await wallet.getUsedAddresses();
    const primaryAddress = addresses;

    // Retrieve all assets to find Handle.me NFTs
    const assets = await wallet.getAssets();

    // Filter for Handle.me NFTs (policy ID based filtering)
    const handles = assets.filter(asset =>
      asset.policyId ===
'f0ff48bbb7bbe9d59a40f1ce90e9e9d0ff5002ec48f232b49ca0fb9a' //
Handle.me policy ID
    );

    return {
      wallet,
      address: primaryAddress,
      handles: handles
    };
  } catch (error) {
    throw new Error(`Wallet connection failed:
${error.message}`);
  }
};
```

The Handle.me integration requires understanding that each handle is an NFT with specific metadata that resolves to wallet addresses. Companies need a main handle (like

$agritech_corp) that serves as the root identity, with sub-handles created for different roles and individuals. The sub-handle system enables hierarchical identity management where the company maintains control while delegating verification authority to specific employees.

For React applications, MeshSDK provides hooks that simplify wallet integration:

```typescript
import { useWallet, useAddress, useAssets } from
'@meshsdk/react';

const WalletConnector = () => {
  const { wallet, connect, connected } = useWallet();
  const address = useAddress();
  const assets = useAssets();

  const companyHandles = assets?.filter(asset =>
    asset.policyId ===
'f0ff48bbb7bbe9d59a40f1ce90e9e9d0ff5002ec48f232b49ca0fb9a' &&
    asset.assetName.includes('agritech_corp') // Company's handle
pattern
  );

  return (
    <div>
      {!connected ? (
        <button onClick={() => connect('eternl')}>
          Connect Wallet
        </button>
      ) : (
        <div>
          <p>Connected: {address}</p>
          <p>Company Handles: {companyHandles?.length || 0}</p>
        </div>
      )}
    </div>
  );
};
```

Administrative System Implementation

The administrative system manages the relationship between company handles and employee identities. Companies start by acquiring a main Handle.me NFT that represents their organizational identity. This handle serves as the root for all sub-handles created for employees, creating a verifiable hierarchy that connects individual actions to organizational authorization.

Database design for the administrative system should separate public blockchain data from private employee information. The blockchain contains only the Handle.me NFT references and verification timestamps, while employee details remain in traditional databases with appropriate access controls.

```typescript
// Database schema example (using PostgreSQL/Supabase)
const employeeSchema = {
  id: 'uuid PRIMARY KEY',
  email: 'varchar UNIQUE NOT NULL',
  full_name: 'varchar NOT NULL',
  role: 'varchar NOT NULL', // field_supervisor,
quality_inspector, etc.
  handle_nft_id: 'varchar UNIQUE', // Reference to Handle.me
sub-handle
  biometric_hash: 'varchar', // Hashed biometric template
  created_at: 'timestamp',
  verified_at: 'timestamp',
  active: 'boolean DEFAULT true'
};

const companyHandleSchema = {
  id: 'uuid PRIMARY KEY',
  main_handle: 'varchar UNIQUE NOT NULL', // $agritech_corp
  policy_id: 'varchar NOT NULL',
  asset_name: 'varchar NOT NULL',
  wallet_address: 'varchar NOT NULL',
  created_at: 'timestamp'
};
```

The administrative interface handles employee onboarding by creating database records and associating them with Handle.me sub-handles. The system should verify that the company

wallet owns the main handle before allowing sub-handle assignments. This prevents unauthorized organizations from claiming company affiliation.

For sub-handle management, companies can use either NFT-based sub-handles (which exist as separate NFTs) or virtual sub-handles (which are managed through the Handle.me registry). Virtual sub-handles provide more flexibility for employee management since they don't require separate NFT transactions for each modification.

TypeScript

```
const assignEmployeeHandle = async (employeeId, subHandleName) =>
{
  // Verify company owns main handle
  const companyHandle = await getCompanyMainHandle();
  if (!companyHandle) {
    throw new Error('Company main handle not verified');
  }

  // Create sub-handle association
  const subHandle =
`${subHandleName}@${companyHandle.main_handle}`;

  // Update employee record
  await updateEmployee(employeeId, {
    handle_nft_id: subHandle,
    verified_at: new Date()
  });

  return subHandle;
};
```

Biometric Verification Integration

Winter Auth's biometric verification ensures that only authorized individuals can use assigned handles to submit supply chain data. The implementation should capture biometric templates during employee onboarding and store them securely for later verification. The system can store raw biometric images or, for a more secure approach, only mathematical templates are stored that can't be reverse-engineered to recreate original biometric data.

JavaScript

```
import { createWinterAuth } from '@zengate/winter-auth';

const winterAuth = createWinterAuth({
  mode: 'direct', // Server-side implementation
  enableConsoleLogging: true,
  securityLevel: 'MEDIUM'
});

// Initialize Winter Auth
await winterAuth.initialize();

// Employee enrollment
const enrollEmployee = async (employeeId, imageFile) => {
  try {
    // Verify image quality first
    const qualityCheck = await
winterAuth.checkImageQuality(imageFile);

    if (!qualityCheck.isGoodQuality) {
      throw new Error(`Image quality insufficient:
${qualityCheck.issues.join(', ')}`);
    }

    // Store biometric template (server-side only)
    const biometricTemplate = await
generateBiometricTemplate(imageFile);
    const templateHash = await hashTemplate(biometricTemplate);

    // Update employee record with hashed template
```

```
    await updateEmployee(employeeId, {
      biometric_hash: templateHash,
      enrollment_completed: true
    });

    return {
      success: true,
      templateId: templateHash
    };

  } catch (error) {
    console.error('Enrollment failed:', error);
    throw error;
  }
};
```

The verification process occurs when employees attempt to submit supply chain data. The system captures a live image, verifies it against the stored template, and only proceeds with data submission if verification succeeds. This creates an audit trail linking every supply chain record to a verified individual.

JavaScript

```
// Field verification before data submission
const verifyAndSubmit = async (employeeHandle, capturedImage,
supplyChainData) => {
  // Get employee record by handle
  const employee = await getEmployeeByHandle(employeeHandle);
  if (!employee || !employee.active) {
    throw new Error('Employee not found or inactive');
  }

  // Verify captured image against stored template
  const verificationResult = await winterAuth.compareByImage(
    employee.biometric_template, // Retrieved from secure storage
    capturedImage
```

```
  );

  if (!verificationResult.verified) {
    throw new Error('Biometric verification failed');
  }

  // Proceed with Winter Protocol data submission
  const blockchainRecord = await submitToWinterProtocol({
    data: supplyChainData,
    verifiedBy: employeeHandle,
    verificationTimestamp: new Date(),
    verificationConfidence: verificationResult.confidence
  });

  return {
    success: true,
    transactionId: blockchainRecord.txHash,
    verifiedBy: employeeHandle
  };
};
```

Winter Protocol Integration

Winter Protocol integration creates the immutable supply chain records that include verified identity information. Each record contains not just the supply chain data (harvest weights, quality grades, GPS coordinates), but also cryptographic proof of who verified that information and when.

The integration should format data according to EPCIS standards while including the Handle.me identity references that enable auditor verification. This ensures compatibility with existing supply chain systems while adding the identity verification capabilities that regulatory compliance requires, since Winter Protocol is modular, it can be changed to only enable specific templates, or, a more open json structure format, based on the need of the standard implemented.

JavaScript

```
// Winter Protocol data submission
const submitToWinterProtocol = async ({
  data,
  verifiedBy,
  verificationTimestamp,
  verificationConfidence
}) => {
  // Format data according to EPCIS standards
  const epcisEvent = {
    eventType: 'ObjectEvent',
    eventTime: verificationTimestamp.toISOString(),
    eventTimeZoneOffset: '+00:00',
    epcList: data.epcList || [],
    action: 'OBSERVE',
    bizStep: data.bizStep,
    disposition: data.disposition,
    bizLocation: data.location,

    // Winter Auth verification extension
    winterAuth: {
      verifiedBy: verifiedBy, // Handle.me sub-handle
      verificationMethod: 'biometric',
      confidence: verificationConfidence,
      timestamp: verificationTimestamp.toISOString()
```

```
    },

    // Custom business data
    customData: data.customFields || {}
  };

  // Submit to Winter Protocol
  const transaction = await
winterProtocol.submitEvent(epcisEvent);

  return {
    txHash: transaction.hash,
    blockHeight: transaction.blockHeight,
    timestamp: transaction.timestamp
  };
};
```

User Interface Implementation

The field application interface guides supervisors through the verification and data submission process. The interface should be simple enough for use in agricultural environments while maintaining the security properties needed for regulatory compliance.

```javascript
// React simple component for field data submission
const FieldSubmissionForm = () => {
  const [employee, setEmployee] = useState(null);
  const [capturedImage, setCapturedImage] = useState(null);
  const [verificationStatus, setVerificationStatus] =
useState('pending');
  const [formData, setFormData] = useState({});

  const handleBiometricCapture = async () => {
    try {
      // Capture image using device camera
      const image = await captureFromCamera();
      setCapturedImage(image);

      // Verify against stored template
      const result = await verifyBiometric(employee.handle,
image);

      if (result.verified) {
        setVerificationStatus('verified');
      } else {
        setVerificationStatus('failed');
        alert('Biometric verification failed. Please try
again.');
      }
    } catch (error) {
      console.error('Verification error:', error);
      setVerificationStatus('error');
    }
  };
```

```
  const handleSubmit = async () => {
    if (verificationStatus !== 'verified') {
      alert('Biometric verification required before submission');
      return;
    }

    try {
      const result = await verifyAndSubmit(
        employee.handle,
        capturedImage,
        formData
      );

      alert(`Data submitted successfully. Transaction:
${result.transactionId}`);

      // Reset form
      setCapturedImage(null);
      setVerificationStatus('pending');
      setFormData({});

    } catch (error) {
      alert(`Submission failed: ${error.message}`);
    }
  };

  return (
    <div className="field-submission-form">
      <h2>Supply Chain Data Submission</h2>

      {/* Employee identification */}
      <div className="employee-section">
        <label>Employee Handle:</label>
        <input
          value={employee?.handle || ''}
```

```
          onChange={(e) => setEmployee({...employee, handle:
e.target.value})}
        />
      </div>

      {/* Biometric verification */}
      <div className="verification-section">
        <button onClick={handleBiometricCapture}>
          Verify Identity
        </button>
        <span className={`status ${verificationStatus}`}>
          {verificationStatus.toUpperCase()}
        </span>
      </div>

      {/* Supply chain data form */}
      <div className="data-section">
        <label>Harvest Weight (kg):</label>
        <input
          type="number"
          value={formData.weight || ''}
          onChange={(e) => setFormData({...formData, weight:
e.target.value})}
        />

        <label>Quality Grade:</label>
        <select
          value={formData.grade || ''}
          onChange={(e) => setFormData({...formData, grade:
e.target.value})}
        >
          <option value="">Select Grade</option>
          <option value="A">Grade A</option>
          <option value="B">Grade B</option>
          <option value="C">Grade C</option>
        </select>
```

```
        <label>GPS Coordinates:</label>
        <input
          value={formData.coordinates || ''}
          onChange={(e) => setFormData({...formData, coordinates:
e.target.value})}
          placeholder="Latitude, Longitude"
        />
      </div>

      {/* Submit button */}
      <button
        onClick={handleSubmit}
        disabled={verificationStatus !== 'verified'}
        className="submit-button"
      >
        Submit to Blockchain
      </button>
    </div>
  );
};
```

Database and User Management Options

The system supports multiple database and user management approaches depending on organizational needs and technical requirements. Firebase provides rapid prototyping with built-in authentication, while Supabase offers more control with PostgreSQL backing. Enterprise organizations might prefer traditional database systems with custom authentication.

Firebase implementation provides real-time synchronization and offline capabilities:

JavaScript

```
// Firebase configuration
import { initializeApp } from 'firebase/app';
import { getFirestore } from 'firebase/firestore';
import { getAuth } from 'firebase/auth';
import { collection, addDoc, query, where, getDocs,
serverTimestamp } from 'firebase/firestore';

const firebaseConfig = {
  apiKey: "YOUR_FIREBASE_API_KEY",
  authDomain: "YOUR_FIREBASE_AUTH_DOMAIN",
  projectId: "YOUR_FIREBASE_PROJECT_ID",
  storageBucket: "YOUR_FIREBASE_STORAGE_BUCKET",
  messagingSenderId: "YOUR_FIREBASE_MESSAGING_SENDER_ID",
  appId: "YOUR_FIREBASE_APP_ID"
};

const app = initializeApp(firebaseConfig);
const db = getFirestore(app);
const auth = getAuth(app);

// Employee management with Firebase
const addEmployee = async (employeeData) => {
  const docRef = await addDoc(collection(db, 'employees'), {
    ...employeeData,
    createdAt: serverTimestamp(),
    active: true
  });
```

```
  return docRef.id;
};

const getEmployeeByHandle = async (handle) => {
  const q = query(
    collection(db, 'employees'),
    where('handle_nft_id', '==', handle),
    where('active', '==', true)
  );
  const querySnapshot = await getDocs(q);
  return querySnapshot.empty ? null : querySnapshot.docs.data();
};
```

Supabase provides more control with SQL databases:

JavaScript

```
// Supabase configuration
import { createClient } from '@supabase/supabase-js';

const supabaseUrl = 'your-supabase-url';
const supabaseKey = 'your-supabase-anon-key';
const supabase = createClient(supabaseUrl, supabaseKey);

// Employee management with Supabase
const addEmployee = async (employeeData) => {
  const { data, error } = await supabase
    .from('employees')
    .insert([employeeData])
    .select();
  if (error) throw error;
  return data;
};

const getEmployeeByHandle = async (handle) => {
  const { data, error } = await supabase
```

```
    .from('employees')
    .select('*')
    .eq('handle_nft_id', handle)
    .eq('active', true)
    .single();
  if (error && error.code !== 'PGRST116') throw error; // Handle
no rows found gracefully
  return data;
};
```

Security Considerations and Best Practices

Implementation security requires careful attention to biometric data handling, private key management, and audit trail integrity. Biometric templates should never be stored in plain text or transmitted over unencrypted channels. The system should use secure hashing for biometric templates and implement proper access controls for sensitive employee data.

Private key management becomes critical since the system needs to sign blockchain transactions on behalf of the organization. Hardware security modules (HSMs) or secure key management services should protect signing keys, especially for production deployments handling valuable agricultural commodities.

Regulatory Compliance and Audit Support

The implementation creates comprehensive audit trails that regulatory bodies can independently verify. Each blockchain record includes not just the supply chain data, but also cryptographic proof of who verified that data and when. This enables auditors to trace any supply chain claim back to a specific, verified individual.

The Handle.me integration provides human-readable identity references that auditors can understand without requiring deep blockchain expertise. Auditors can verify that $john.doe@agritech_corp represents a real employee of the organization and that the biometric verification succeeded at the time of data submission.

Future Enhancements and Integration Opportunities

The foundation established by this implementation enables numerous enhancements that can improve supply chain verification and regulatory compliance. Integration with IoT sensors can provide automatic data collection that still requires human verification for critical checkpoints. Machine learning analysis of biometric verification patterns can detect potential fraud attempts or system abuse.

Integration with existing ERP systems can streamline data flow between traditional business systems and blockchain verification. The modular architecture enables gradual expansion from pilot programs to enterprise-wide deployment without requiring complete system replacement.

The Handle.me integration provides a foundation for broader Cardano ecosystem integration, including DeFi protocols for supply chain financing and NFT systems for product certification. As the Cardano ecosystem develops, organizations implementing this system will be positioned to take advantage of new capabilities while maintaining their existing verification infrastructure.

The Winter Auth and Winter Protocol combination creates a robust foundation for supply chain identity verification that meets current regulatory requirements while providing flexibility for future enhancements. The system proves that blockchain technology can solve real business problems when implemented with proper attention to security, usability, and regulatory compliance.

Companies implementing this system gain more than just regulatory compliance—they build digital infrastructure that enables new business models, reduces verification costs, and creates competitive advantages in increasingly complex global supply chains. The question isn't whether to implement blockchain-based identity verification, but how quickly organizations can deploy these capabilities to meet growing regulatory demands and market expectations.