# Secure Client-Side Deduplication in AWS - Implementation Guide

## Table of Contents

## Introduction

This documentation provides a comprehensive guide to implementing a secure client-side deduplication system in AWS. The solution reduces storage costs by eliminating redundant data while maintaining strong security guarantees through cryptographic techniques.

## System Architecture

### High-Level Design

The system consists of three main components:

1. Client Application
   - File chunking and hashing
   - Proof generation
   - Encryption/decryption

2. Deduplication Service (AWS)
   - Lambda functions for verification
   - S3 for encrypted storage
   - DynamoDB for metadata

3. Security Layer
   - IAM for access control
   - KMS for key management
   - Proof-of-ownership protocol

### Data Flow

1. Client computes file hash
2. System checks for existing duplicates
3. If new file, client generates proof and uploads encrypted data
4. If duplicate, client proves ownership to gain access

## Setup and Configuration

### Prerequisites

- AWS Account with Admin privileges
- Node.js 16.x+

- AWS CLI v2
- Terraform 1.0+

### Environment Setup

1. AWS Configuration
```bash
aws configure
# Enter your AWS Access Key, Secret Key, and default region
```

2. Install Dependencies
```bash
npm install
cd terraform && terraform init
```

3. Environment Variables
Create `.env` file with:
```env
AWS_REGION=us-east-1
S3_BUCKET=your-deduplication-bucket
DYNAMODB_TABLE=file_metadata
KMS_KEY_ID=alias/deduplication-key
```

## Implementation Details

### 1. File Processing (Client Side)
```javascript
// File chunking and hashing
async function processFile(file) {
  const chunkSize = 1024 * 1024; // 1MB chunks
  const chunks = [];
  const hashes = [];

  for (let i = 0; i < file.size; i += chunkSize) {
    const chunk = file.slice(i, i + chunkSize);
    const hash = await crypto.subtle.digest('SHA-256', chunk);
    hashes.push(bufferToHex(hash));
    chunks.push(chunk);
  }

  return { chunks, hashes };
}
```

### 2. Proof-of-Ownership Protocol

The protocol implements a challenge-response mechanism:

1. Server stores H(file) and H(H(file) + salt)
2. For verification:
   - Server sends random challenge C

   - Client computes H(H(file) + C)
   - Server verifies the response

### 3. AWS Infrastructure (Terraform)
```hcl
# S3 Bucket for encrypted storage
resource "aws_s3_bucket" "dedupe_storage" {
  bucket = "secure-deduplication-data"

  server_side_encryption_configuration {
    rule {
      apply_server_side_encryption_by_default {
        kms_master_key_id = aws_kms_key.dedupe_key.arn
        sse_algorithm     = "aws:kms"
      }
    }
  }
}

# Deduplication Lambda Function
resource "aws_lambda_function" "dedupe_check" {
  filename      = "lambda/dedupe_check.zip"
  function_name = "dedupe-check"
  role          = aws_iam_role.lambda_exec.arn
  handler       = "index.handler"
  runtime       = "nodejs16.x"

  environment {
    variables = {
      METADATA_TABLE = aws_dynamodb_table.metadata.name
    }
  }
}
```

## Security Considerations

### Key Management

1. Use AWS KMS for encryption keys
2. Implement key rotation policies
3. Store content-derived keys encrypted with KMS

### Access Control
```json
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "s3:PutObject",
        "s3:GetObject"
      ],
      "Resource": "arn:aws:s3:::secure-deduplication-data/*",
      "Condition": {
        "StringEquals": {
          "s3:x-amz-server-side-encryption": "aws:kms"
        }
      }
    }
  ]
}
```

## Testing and Validation

### Test Cases

1. Duplicate Detection
   - Upload same file twice, verify only one copy stored
   - Verify hash collision handling

2. Proof-of-Ownership
   - Test with valid and invalid proofs
   - Measure verification time

3. Performance Testing
   - Benchmark with files of varying sizes
   - Test concurrent uploads

### Test Script
```bash
npm test
# Runs:
# - Unit tests
# - Integration tests (requires AWS environment)
# - Security tests
```

## Deployment Guide

### Production Deployment
```bash
cd terraform
terraform plan -out deployment.plan
terraform apply deployment.plan
```

```bash
cd ../lambda
./deploy.sh --production
```

```bash
npm run build
```

### CI/CD Pipeline
```yaml
name: Deploy
on:
  push:
    branches: [ main ]

jobs:
  deploy:
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@v2
      - uses: actions/setup-node@v2
        with:
          node-version: '16'
      - run: npm install
      - run: npm test
      - uses: hashicorp/setup-terraform@v1
      - run: cd terraform && terraform apply -auto-approve
```

## Maintenance and Monitoring

### CloudWatch Alarms

Set up alarms for:
- Failed deduplication attempts
- Unauthorized access attempts
- Storage capacity thresholds

### Logging Configuration
```javascript
const AWS = require('aws-sdk');
const cloudwatch = new AWS.CloudWatchLogs();

async function logEvent(event) {
  await cloudwatch.putLogEvents({
    logGroupName: '/aws/lambda/dedupe-check',
    logStreamName: 'proof-verification',
    logEvents: [{
      message: JSON.stringify(event),
      timestamp: Date.now()
    }]
  }).promise();
}
```

## Troubleshooting

### Common Issues

1. Permission Denied Errors
   - Verify IAM roles and policies
   - Check KMS key permissions

2. Duplicate Detection Failures
   - Verify hash computation matches on client/server
   - Check DynamoDB consistency settings

3. Performance Bottlenecks
   - Review Lambda memory allocation
   - Check for DynamoDB throttling

### Debugging Tools
```bash
aws logs tail /aws/lambda/dedupe-check --follow
aws s3api list-objects --bucket secure-deduplication-data
```

## References

1. AWS Security Best Practices
2. "Proofs of Ownership in Cloud Storage" - Cryptology ePrint Archive
3. Terraform AWS Provider Documentation
4. Node.js Crypto Module Documentation