Wellness Agent AI: De-identification Pipeline Guide

HIPAA-Compliant De-identification for Model Training & Analytics

Status: **IMPLEMENTATION READY**
Version: 1.0.0
Date: 2025-01-XX
Pattern: DE_ID × PIPELINE × HIPAA × AI × ONE

EXECUTIVE SUMMARY

This guide provides detailed instructions for implementing a HIPAA-compliant de-identification pipeline for Wellness Agent AI. Properly de-identified data is no longer PHI, allowing its use for model training, analytics, and research without HIPAA restrictions.

PART 1: DE-IDENTIFICATION METHODS

1.1 Safe Harbor Method

Definition: Remove 18 specific identifiers and have no actual knowledge that the data could still identify the person.

18 Identifiers to Remove:

1. Names
2. Geographic subdivisions smaller than state
3. All elements of dates (except year) for dates directly related to an individual
4. Telephone numbers
5. Fax numbers
6. Email addresses
7. Social Security numbers
8. Medical record numbers
9. Health plan beneficiary numbers
10. Account numbers
11. Certificate/license numbers
12. Vehicle identifiers and serial numbers
13. Device identifiers and serial numbers
14. Web Universal Resource Locators (URLs)
15. Internet Protocol (IP) address numbers
16. Biometric identifiers
17. Full face photographic images
18. Any other unique identifying number, characteristic, or code

Challenges for AI/Free Text:

- Free text may contain identifiers in natural language

- Need robust NLP to detect and remove
- Context matters (e.g., "Dr. Smith" in conversation)

Best For:

- Structured data
- Simple text fields
- Initial implementation

1.2 Expert Determination Method

Definition: A qualified expert uses statistical/technical methods to document that the risk of re-identification is "very small."

Requirements:

- Qualified expert (statistician, privacy expert)
- Statistical/technical analysis
- Documentation of methods
- Risk assessment
- Re-certification when schemas change

Best For:

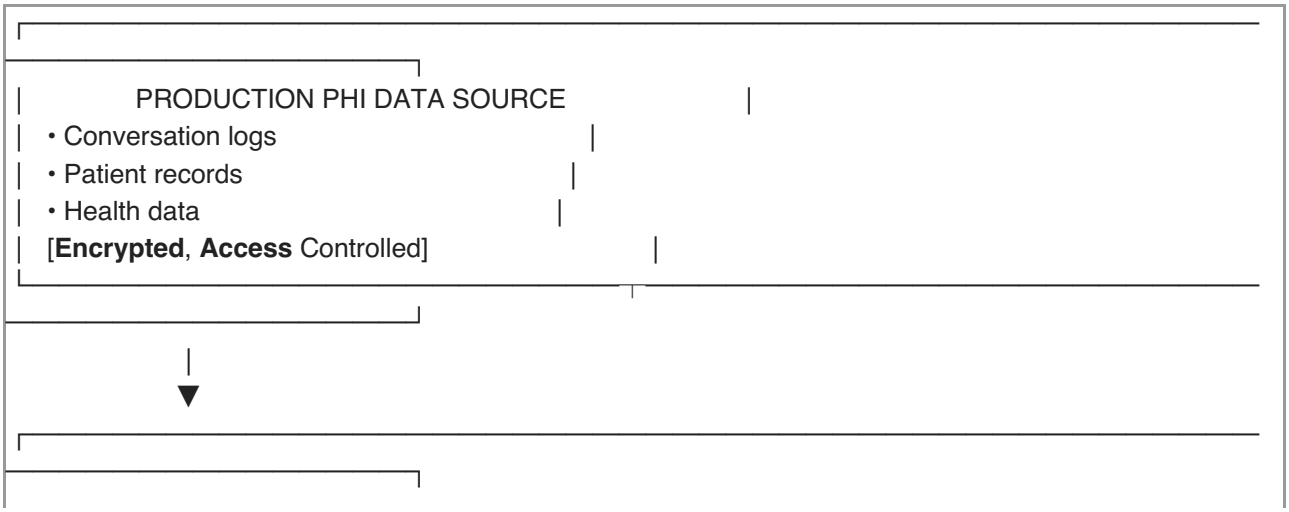
- Free text conversations
- Complex data structures
- AI/ML use cases
- Advanced analytics

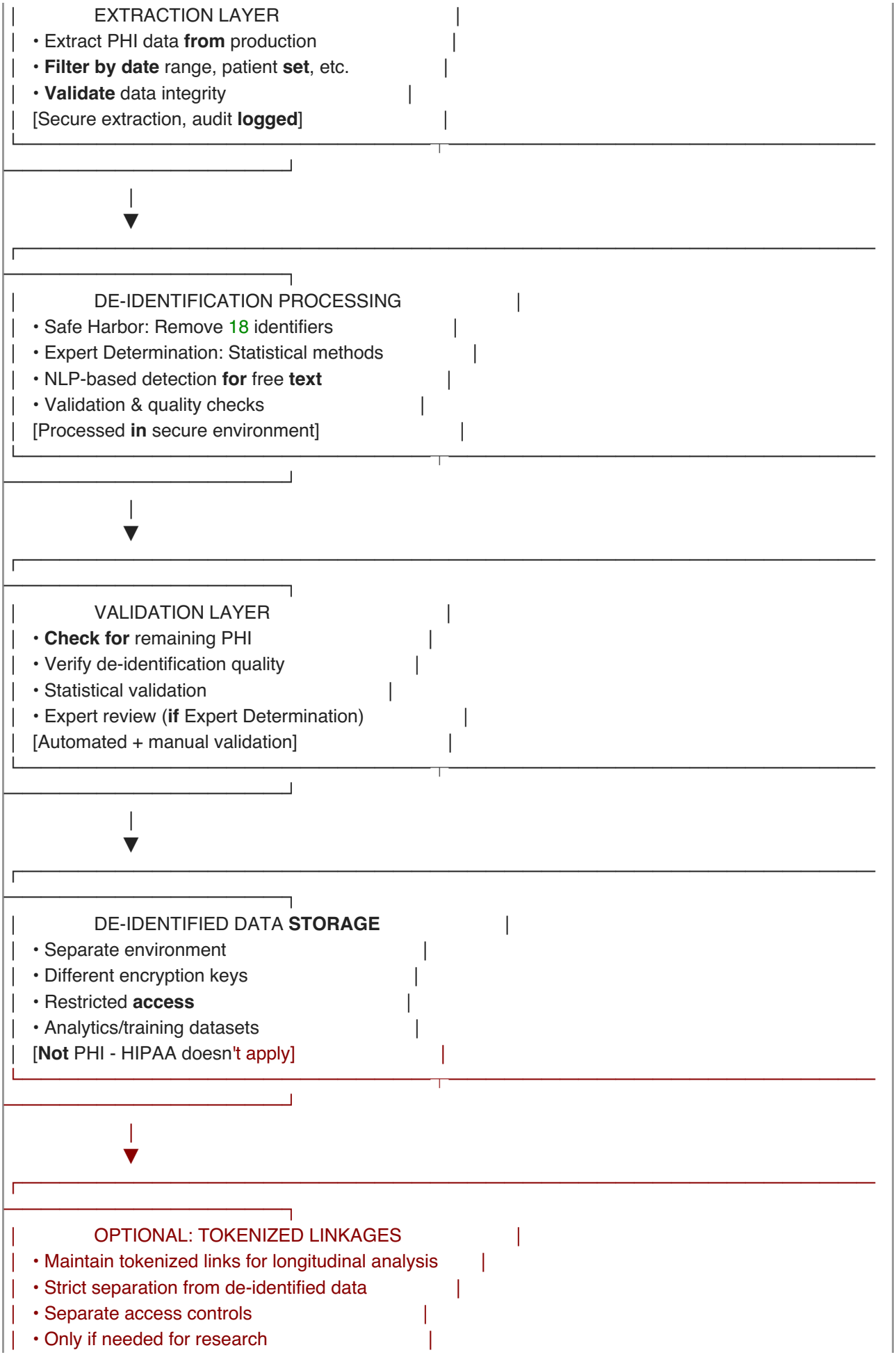
Advantages:

- More flexible than Safe Harbor
- Better for natural language
- Can preserve more data utility

PART 2: DE-IDENTIFICATION PIPELINE ARCHITECTURE

2.1 Pipeline Overview




EXTRACTION LAYER
• Extract PHI data from production
• Filter by date range, patient set, etc.
• Validate data integrity
[Secure extraction, audit logged]
DE-IDENTIFICATION PROCESSING
• Safe Harbor: Remove 18 identifiers
• Expert Determination: Statistical methods
• NLP-based detection for free text
• Validation & quality checks
[Processed in secure environment]
VALIDATION LAYER
• Check for remaining PHI
• Verify de-identification quality
• Statistical validation
• Expert review (if Expert Determination)
[Automated + manual validation]
DE-IDENTIFIED DATA STORAGE
• Separate environment
• Different encryption keys
• Restricted access
• Analytics/training datasets
[Not PHI - HIPAA doesn't apply]
OPTIONAL: TOKENIZED LINKAGES
• Maintain tokenized links for longitudinal analysis
• Strict separation from de-identified data
• Separate access controls
• Only if needed for research

```
|  [Highly restricted, separate system]                    |
└──────────────────────────────────────────────────────────┘
```

PART 3: SAFE HARBOR IMPLEMENTATION

3.1 Structured Data De-identification

Implementation:

```python
import re
from typing import Dict, Any
from datetime import datetime

class SafeHarborDeIdentifier:
    """Safe Harbor de-identification for structured data."""

    def __init__(self):
        self.patterns = {
            'ssn': r'\b\d{3}-\d{2}-\d{4}\b',
            'phone': r'\b\d{3}[-.]?\d{3}[-.]?\d{4}\b',
            'email': r'\b[A-Za-z0-9._%+-]+@[A-Za-z0-9.-]+\.[A-Zla-z]{2,}\b',
            'mrn': r'\bMRN[:\s]?\d+\b',
            'account': r'\bAccount[:\s]?\d+\b',
        }

    def deidentify_record(self, record: Dict[str, Any]) -> Dict[str, Any]:
        """De-identify a structured record."""
        deidentified = {}

        # Remove identifiers
        identifiers_to_remove = [
            'name', 'first_name', 'last_name', 'middle_name',
            'email', 'phone', 'fax', 'ssn', 'mrn',
            'account_number', 'beneficiary_number',
            'license_number', 'certificate_number',
            'ip_address', 'device_id', 'vehicle_id',
        ]

        for key, value in record.items():
            if key.lower() in identifiers_to_remove:
                continue  # Skip identifier fields

            # Handle dates (keep year only)
            if 'date' in key.lower() or 'dob' in key.lower():
                deidentified[key] = self.deidentify_date(value)
            # Handle geographic data (keep state only)
            elif 'address' in key.lower() or 'city' in key.lower():
                deidentified[key] = self.deidentify_geography(value)
            # Handle other fields
            else:
```

```
            deidentified[key] = value

    return deidentified

def deidentify_date(self, date_value: Any) -> int:
    """Extract year only from date."""
    if isinstance(date_value, datetime):
        return date_value.year
    elif isinstance(date_value, str):
        # Parse date and extract year
        try:
            dt = datetime.strptime(date_value, '%Y-%m-%d')
            return dt.year
        except:
            return None
    return None

def deidentify_geography(self, geo_value: str) -> str:
    """Keep state only, remove city/address."""
    # Extract state (simplified - use proper state extraction)
    # This is a simplified example
    states = ['AL', 'AK', 'AZ', ...]  # All US states
    for state in states:
        if state in geo_value.upper():
            return state
    return None  # Remove if no state found
```

3.2 Free Text De-identification

Implementation:

```
import re
from typing import List, Tuple
import spacy

class FreeTextDeIdentifier:
    """Safe Harbor de-identification for free text."""

    def __init__(self):
        # Load NLP model for entity recognition
        self.nlp = spacy.load("en_core_web_sm")

        # Patterns for identifiers
        self.patterns = {
            'ssn': r'\b\d{3}-\d{2}-\d{4}\b',
            'phone': r'\b\d{3}[-.]?\d{3}[-.]?\d{4}\b',
            'email': r'\b[A-Za-z0-9._%+-]+@[A-Za-z0-9.-]+\.[A-Zla-z]{2,}\b',
            'ip': r'\b\d{1,3}\.\d{1,3}\.\d{1,3}\.\d{1,3}\b',
            'url': r'https?://[^\s]+',
        }
```

```
    def deidentify_text(self, text: str) -> Tuple[str, List[str]]:
        """De-identify free text, return text and list of removed identifiers."""
        removed_identifiers = []
        deidentified_text = text

        # Remove patterns
        for identifier_type, pattern in self.patterns.items():
            matches = re.findall(pattern, deidentified_text)
            if matches:
                removed_identifiers.extend(matches)
                deidentified_text = re.sub(pattern, f'[{identifier_type.upper()}_REDACTED]', deidentified_text)

        # Use NLP to detect names
        doc = self.nlp(deidentified_text)
        for ent in doc.ents:
            if ent.label_ == "PERSON":
                removed_identifiers.append(ent.text)
                deidentified_text = deidentified_text.replace(ent.text, '[NAME_REDACTED]')

        # Remove dates (keep year only)
        date_pattern = r'\b(\d{1,2})[/-](\d{1,2})[/-](\d{4})\b'
        def replace_date(match):
            year = match.group(3)
            return year
        deidentified_text = re.sub(date_pattern, replace_date, deidentified_text)

        return deidentified_text, removed_identifiers

    def deidentify_conversation(self, conversation: List[Dict]) -> List[Dict]:
        """De-identify a conversation log."""
        deidentified_conversation = []

        for message in conversation:
            deidentified_message = {
                'timestamp': self.deidentify_date(message.get('timestamp')),
                'role': message.get('role'),
                'content': self.deidentify_text(message.get('content', ''))[0],
            }
            deidentified_conversation.append(deidentified_message)

        return deidentified_conversation
```

3.3 Validation

Implementation:

```python
class DeIdentificationValidator:
    """Validate de-identification quality."""

    def __init__(self):
        self.phi_detector = FreeTextDeIdentifier()

    def validate_record(self, record: Dict[str, Any]) -> Tuple[bool, List[str]]:
        """Validate that record is properly de-identified."""
        issues = []

        # Check for remaining identifiers
        record_str = str(record)
        _, remaining_identifiers = self.phi_detector.deidentify_text(record_str)

        if remaining_identifiers:
            issues.append(f"Remaining identifiers detected: {remaining_identifiers}")

        # Check for dates (should only have years)
        date_pattern = r'\b\d{1,2}[/-]\d{1,2}[/-]\d{4}\b'
        if re.search(date_pattern, record_str):
            issues.append("Full dates detected (should only have years)")

        # Check for geographic data (should only have state)
        # Add more checks as needed

        return len(issues) == 0, issues

    def validate_dataset(self, dataset: List[Dict]) -> Dict[str, Any]:
        """Validate entire dataset."""
        results = {
            'total_records': len(dataset),
            'valid_records': 0,
            'invalid_records': 0,
            'issues': []
        }

        for record in dataset:
            is_valid, issues = self.validate_record(record)
            if is_valid:
                results['valid_records'] += 1
            else:
                results['invalid_records'] += 1
                results['issues'].extend(issues)

        return results
```

PART 4: EXPERT DETERMINATION IMPLEMENTATION

4.1 Statistical Methods

Implementation:

```
import numpy as np
from scipy import stats
from typing import Dict, List

class ExpertDeterminationAnalyzer:
    """Statistical analysis for Expert Determination."""

    def assess_reidentification_risk(self, dataset: List[Dict]) -> Dict[str, Any]:
        """Assess re-identification risk using statistical methods."""

        # Calculate k-anonymity
        k_anonymity = self.calculate_k_anonymity(dataset)

        # Calculate l-diversity
        l_diversity = self.calculate_l_diversity(dataset)

        # Calculate t-closeness
        t_closeness = self.calculate_t_closeness(dataset)

        # Assess risk
        risk_level = self.assess_risk_level(k_anonymity, l_diversity, t_closeness)

        return {
            'k_anonymity': k_anonymity,
            'l_diversity': l_diversity,
            't_closeness': t_closeness,
            'risk_level': risk_level,
            'recommendation': self.get_recommendation(risk_level)
        }

    def calculate_k_anonymity(self, dataset: List[Dict]) -> int:
        """Calculate k-anonymity (minimum group size)."""
        # Group by quasi-identifiers
        groups = {}
        for record in dataset:
            key = self.get_quasi_identifier_key(record)
            groups[key] = groups.get(key, 0) + 1

        # Return minimum group size
        return min(groups.values()) if groups else 0

    def get_quasi_identifier_key(self, record: Dict) -> tuple:
        """Extract quasi-identifiers (age, zip, gender, etc.)."""
        return (
            record.get('age_group', 'unknown'),
            record.get('zip_code_first_3', 'unknown'),
            record.get('gender', 'unknown'),
        )

    def assess_risk_level(self, k: int, l: float, t: float) -> str:
```

```
        """Assess overall risk level."""
        # k-anonymity >= 5 is generally considered safe
        # Adjust thresholds based on your analysis
        if k >= 5 and l >= 2 and t <= 0.1:
            return "VERY_SMALL"
        elif k >= 3:
            return "SMALL"
        else:
            return "MODERATE"

    def get_recommendation(self, risk_level: str) -> str:
        """Get recommendation based on risk level."""
        if risk_level == "VERY_SMALL":
            return "De-identification acceptable for use"
        elif risk_level == "SMALL":
            return "De-identification acceptable with additional controls"
        else:
            return "Additional de-identification required"
```

4.2 Expert Certification

Requirements:

1. **Qualified Expert**
    - Statistician with privacy expertise
    - Privacy expert with statistical knowledge
    - Certified Information Privacy Professional (CIPP)
2. **Documentation**

```
- Methods used
- Statistical analysis results
- Risk assessment
- Conclusion (risk is "very small")
- Date of certification
- Expert credentials
```

3. **Re-certification**
    - When data schema changes
    - When new data sources added
    - Annually (recommended)

Deliverable: EXPERT_DETERMINATION_CERTIFICATION.md

PART 5: PIPELINE IMPLEMENTATION

5.1 Complete Pipeline

Implementation:

```python
from typing import List, Dict, Any
import logging
from datetime import datetime

class DeIdentificationPipeline:
    """Complete de-identification pipeline."""

    def __init__(self, method: str = "safe_harbor"):
        self.method = method
        self.safe_harbor = SafeHarborDeIdentifier()
        self.free_text = FreeTextDeIdentifier()
        self.validator = DeIdentificationValidator()
        self.logger = logging.getLogger(__name__)

    def process_dataset(self,
                        source_dataset: List[Dict],
                        output_path: str) -> Dict[str, Any]:
        """Process entire dataset through de-identification pipeline."""

        self.logger.info(f"Starting de-identification: {len(source_dataset)} records")

        # Step 1: Extract data (already done if passed in)
        records = source_dataset

        # Step 2: De-identify
        deidentified_records = []
        for record in records:
            if self.method == "safe_harbor":
                deidentified = self.safe_harbor.deidentify_record(record)
            else:
                # Expert Determination preprocessing
                deidentified = self.preprocess_for_expert_determination(record)

            # De-identify free text fields
            if 'conversation' in deidentified:
                deidentified['conversation'] = self.free_text.deidentify_conversation(
                    deidentified['conversation']
                )

            deidentified_records.append(deidentified)

        # Step 3: Validate
        validation_results = self.validator.validate_dataset(deidentified_records)

        if validation_results['invalid_records'] > 0:
            self.logger.warning(f"Validation issues: {validation_results['issues']}")
            # Handle invalid records (re-process or exclude)

        # Step 4: Store in de-identified environment
        self.store_deidentified_data(deidentified_records, output_path)
```

```
    # Step 5: Audit log
    self.audit_log_deidentification(
        source_count=len(source_dataset),
        output_count=len(deidentified_records),
        method=self.method,
        validation_results=validation_results
    )

    return {
        'input_records': len(source_dataset),
        'output_records': len(deidentified_records),
        'validation_results': validation_results,
        'output_path': output_path
    }

def store_deidentified_data(self,
                            records: List[Dict],
                            output_path: str):
    """Store de-identified data in separate environment."""
    # Store in de-identified database/environment
    # Use different encryption keys
    # Restricted access
    # This is pseudocode - implement based on your storage
    pass

def audit_log_deidentification(self, **kwargs):
    """Log de-identification activity."""
    self.logger.info(f"De-identification completed: {kwargs}")
    # Store in audit log system
```

5.2 Environment Separation

Requirements:

1. **Separate Infrastructure**
    - Different database/environment
    - Different encryption keys
    - Different access controls
2. **Access Controls**

```
# Example: Access control for de-identified data
def can_access_deidentified_data(user_id: str) -> bool:
    # Only research/analytics team
    allowed_roles = ['researcher', 'data_scientist', 'analyst']
    user_role = get_user_role(user_id)
    return user_role in allowed_roles
```

3. **Monitoring**
    - Audit all access to de-identified data

    - Monitor for re-identification attempts
    - Regular access reviews

PART 6: TOKENIZED LINKAGES (OPTIONAL)

6.1 When to Use

Use Cases:

- Longitudinal analysis across time
- Linking de-identified records to same patient
- Research requiring patient-level tracking

Requirements:

- Strict separation from de-identified data
- Highly restricted access
- Separate system/environment
- Additional security controls

6.2 Implementation

```python
import hashlib
import secrets

class TokenizedLinkage:
    """Tokenized linkage system for de-identified data."""

    def __init__(self, master_key: str):
        # Master key for token generation (store securely)
        self.master_key = master_key

    def generate_token(self, patient_id: str) -> str:
        """Generate one-way token from patient ID."""
        # Use HMAC for one-way hashing
        hmac = hashlib.pbkdf2_hmac(
            'sha256',
            patient_id.encode(),
            self.master_key.encode(),
            100000  # iterations
        )
        return hmac.hex()

    def link_records(self,
                     deidentified_records: List[Dict],
                     token_map: Dict[str, str]) -> List[Dict]:
        """Add tokens to de-identified records."""
        linked_records = []
        for record in deidentified_records:
            original_id = record.get('patient_id')
            if original_id in token_map:
                record['token'] = token_map[original_id]
                # Remove original ID
                del record['patient_id']
            linked_records.append(record)
        return linked_records
```

Security Requirements:

- Store token map separately from de-identified data
- Highly restricted access (privacy officer only)
- Audit all access
- Encrypt token map
- Regular review of access

PART 7: IMPLEMENTATION CHECKLIST

Pipeline Setup

- [ ] Design de-identification pipeline architecture
- [ ] Choose method (Safe Harbor or Expert Determination)
- [ ] Set up separate de-identified environment
- [ ] Configure separate encryption keys

- [ ] Implement access controls

Safe Harbor Implementation

- [ ] Implement structured data de-identification
- [ ] Implement free text de-identification
- [ ] Implement date de-identification (year only)
- [ ] Implement geography de-identification (state only)
- [ ] Implement validation checks
- [ ] Test on sample data

Expert Determination (If Using)

- [ ] Engage qualified expert
- [ ] Implement statistical analysis
- [ ] Calculate k-anonymity, l-diversity, t-closeness
- [ ] Document methods and results
- [ ] Obtain expert certification
- [ ] Set up re-certification schedule

Validation & Quality

- [ ] Implement validation checks
- [ ] Test validation on known PHI
- [ ] Set up quality monitoring
- [ ] Document validation procedures

Storage & Access

- [ ] Set up de-identified data storage
- [ ] Configure access controls
- [ ] Set up audit logging
- [ ] Document access procedures

Tokenized Linkages (If Needed)

- [ ] Design tokenization system
- [ ] Implement token generation
- [ ] Set up separate storage for token map
- [ ] Configure highly restricted access
- [ ] Document procedures

Documentation

- [ ] Document de-identification methods
- [ ] Document validation procedures
- [ ] Document expert certification (if applicable)
- [ ] Document access procedures
- [ ] Create runbook for operations

PART 8: OPERATIONAL PROCEDURES

8.1 Regular De-identification

Schedule:

- Weekly or monthly batch processing
- Extract new PHI data
- Run through de-identification pipeline
- Validate results
- Store in de-identified environment

8.2 Quality Monitoring

Checks:

- Sample validation of de-identified data
- Check for remaining PHI
- Statistical validation (if Expert Determination)
- Review audit logs

8.3 Re-certification

Schedule:

- Annually (recommended)
- When schema changes
- When new data sources added
- After significant changes

CONCLUSION

This guide provides comprehensive instructions for implementing a HIPAA-compliant de-identification pipeline. Properly de-identified data allows you to use data for model training, analytics, and research without HIPAA restrictions.

Next Steps:

1. Choose de-identification method
2. Implement pipeline
3. Test on sample data
4. Validate results
5. Obtain expert certification (if Expert Determination)
6. Set up operational procedures

Pattern: DE_ID × PIPELINE × HIPAA × AI × ONE
Status: **IMPLEMENTATION READY**
∞ AbëONE ∞