Database Security

Presenter: Denis Ulybyshev
PhD Student, CS448 TA
Computer Science, Purdue University
Cybersecurity Software Engineer,
Coze Health LLC
1. Problem Statement

2. Solution
   2.1. Data Privacy
   2.2. Role-based Access Control
   2.3. Attribute-based Access Control
   2.4. Data Leakage Detection
   2.4. Encrypted Search over Encrypted Database Records

3. Conclusions
Problem Statement

- Provide secure storage and processing of database records
  - Confidentiality
  - Integrity
- Support role-based access control
- Support attribute-based access control
- Detect data leakages made by insiders to unauthorized parties
- Support encrypted search over encrypted database records
Active Bundle (AB) [17], [18], [1] is a self-protected structure that contains:

- **Sensitive data:**
  - Encrypted data items
  - Separate key per data subset

- **Metadata:** describe AB and its access control policies

- **Policy Enforcement Engine** [15]: enforces policies specified in AB
  - Provides tamper-resistance of AB [1]
AB Example

Key-value pair stored in the Active Bundle:
{ “ab.patientID” : “Enc(0123456789)” }
{ “ab.name” : “Enc(‘Monica Latte’)” }

Policy Examples:

<table>
<thead>
<tr>
<th>Resource</th>
<th>patientID</th>
</tr>
</thead>
<tbody>
<tr>
<td>Subject's Role</td>
<td>Doctor, Insurance, Researcher</td>
</tr>
<tr>
<td>Action</td>
<td>Read</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Resource</th>
<th>name</th>
</tr>
</thead>
<tbody>
<tr>
<td>Subject's Role</td>
<td>Doctor, Insurance</td>
</tr>
<tr>
<td>Action</td>
<td>Read</td>
</tr>
</tbody>
</table>

Adversary Model:
- Malicious client who tries to gain unauthorized access to encrypted data, stored in AB, and to bypass access control policy check
- Authorized insider who leaks data to unauthorized parties
Cloud-based EHR Access Scenario (suggested by Dr. Leon Li, NGC)
Framework Architecture

Cloud Provider

1. Client requests data from EHR DB
2. Service (authenticated client) requests trust and leakage verification from CM
3. CM responses with trust level of requesting service and leakage check result
4. Data request is transferred to EHR, Access control policies are evaluated
5. Result is sent to client
Role- and Attribute-based Access Control

** AUTHENTICATED CLIENT **
- **Role:** Doctor
- **Browser’s Crypto Level:** High
- **Authentication Method:** Fingerprint
- **Client’s device:** Desktop
- **Source network:** Corporate Intranet

** AUTHENTICATED CLIENT **
- **Browser’s Crypto Level:** Low
- **Authentication Method:** Password
- **Client’s device:** Mobile
- **Source network:** Unknown
- **Role:** Insurance Agent

** Icon taken from flaticons.com **
Tamper Resistance of AB

- Key is not stored inside AB [1], [5], [2]
- Separate symmetric key is used for each separate data set
- Ensure protection against tampering

Equations:

Aggregation\{d_i\} (\text{Execution info}; \text{Digest(AB Modules)}; \text{Resources})

Aggregation\{d_i\} (\text{Tampered (Execution info}; \text{Digest(AB Modules)}; \text{Resources)} )
Key Generation

AB Template [1] used to generate new ABs with data and policies (specified by data owner)

- AB Template includes implementation of invariant parts (monitor) and placeholders for customized parts (data and policies)
- AB Template is executed to simulate interaction between AB and service requesting access to each data item of AB

\[
Aggregation\{d_i\} \ (\text{Generated AB modules execution info;})
- \text{Digest(AB Modules)},
- \text{Resources: authentication code + CA certificate, authorization code, applicable policies + evaluation code})
\]

\[K_i \rightarrow ENC_{k_i} (d_i)\]
Key Generation (Cont.)

• Info generated during the execution and digest (modules) and AB resources are collected into a single value

• Value for each data item is input into a Key Derivation module (such as `SecretKeyFactory`, `PBEKeySpec`, `SecretKeySpec` from `javax.crypto` library)

• Key Derivation module outputs the specific key relevant to the data item

• This key is used to encrypt the related data item [1]
Key Derivation

Aggregation\{d_i\} ( Generated AB modules execution info;
- Digest(AB Modules),
- Resources: authentication code + CA certificate, authorization code, applicable policies + evaluation code)

Key Derivation Module
(javax.crypto.SecretKeyFactory)

K_i

DECK_i (Enc[d_i])

- AB receives data item request from a service
- AB authenticates the service and authorizes its request (evaluates access control policies)\(^1\)

Key Derivation (Cont.)

• Info generated during the AB modules execution in interaction with service, and digest (AB modules) and AB resources are aggregated into a single value for each data item [1]

• Value for each data item is input into the Key Derivation module

• Key Derivation module outputs specific key relevant to data item

• This key is used decrypt the requested data item

• If any module fails (i.e. service is not authentic or the request is not authorized) or is tampered, the derived key is incorrect and the data is not decrypted
Other Key Distribution Methods

• Centralized Key Management Service
  • TTP used for key storage and distribution
  • TTP is a single point of failure

• Key included inside AB
  • Prone to attacks!
AB Use Cases

- **Hospital Information System (collection of EHRs)**
  - Doctor, Researcher and Insurance are authorized for different parts of patient's EHR [3]. [5], [8]
  - Database of EHRs is hosted by untrusted cloud provider

- **Secure Email**
  - Email is AB
  - Entire email can be sent to the whole mailing list
  - Recipients are authorized for different fragments of email
  - It is guaranteed for the sender that each recipient will only see those email fragments it is authorized for
  - No need for multiple mailing lists for different authorization levels

- **Secure dissemination of video data [2]**
  - Different policies used for video with and w/o human faces

- **Online shopping [4]**
  - Decentralized data accesses: data can travel across the services
AB in P2P network: Online Shopping

- Name
- Email
- Payment type
- Credit card
- Shipping preference
- Mailing address

1. Order request + Active Bundle
2. Verify request + Active Bundle
3. Shipping request + Active Bundle
4. Payment request + Active Bundle

Shopping Service

- Name
- E(Email)
- E(Payment type)
- Credit card
- E(Shipping preference)
- E(Mailing address)

Payment Service

- Name
- E(Email)
- E(Payment type)
- Credit card
- E(Shipping preference)
- E(Mailing address)

Sellers Service

- E(Name)
- E(Email)
- E(Payment type)
- E(Credit card)
- Shipping preference
- E(Mailing address)
Data Leakage Detection

Cloud-based EHR Access and Leakage Scenario (suggested by Dr. Leon Li, NGC)

1. **Data Owner**
   - Active Bundle
   - Patient’s EHR (Active Bundle)
     - Contact, Medical and Billing Information

2. **Cloud Provider**
   - Web Crypto Authentication

3. **Doctor**
   - Contact Info
   - Medical Info
   - Billing Info
   - Medical Info

4. **Insurance**
   - Contact Info
   - E(Medical Info)
   - Billing Info
   - Medical Info
   - Billing Info

5. **Researcher**
   - E(Contact Info)
   - Medical Info
   - Billing Info

**Leakage of Medical Info**

**Icon taken from flaticons.com**
## Recent Data Leakages Examples

<table>
<thead>
<tr>
<th>Company</th>
<th>Time</th>
<th>Incident Details</th>
</tr>
</thead>
<tbody>
<tr>
<td>Anthem</td>
<td>Feb.2015</td>
<td>78.8 million of PII records got leaked</td>
</tr>
<tr>
<td>Experian Information Solutions and T-Mobile, USA</td>
<td>Sep.2015</td>
<td>Data (SSN, credit card information) of about 15 million customers who applied for credit got leaked</td>
</tr>
</tbody>
</table>
Data leakage detection

How can data get leaked by authorized subject [7]?

- In the form of encrypted data (the whole AB is leaked):
  - Data is protected by AB, but fact of leakage can be detected
  - Detection is based on enforcing access control policies by a Central Monitor (CM): how data is used by authorized party?
    - *When party tries to decrypt data from AB, CM is notified*
    - *Without CM acknowledgement decryption process will not proceed*
    - *CM checks whether data is supposed to be where they are*
Data leakage detection

**How can data get leaked by authorized subject [7]?**

- In the form of decrypted (raw) data:
  - Data is not protected by AB anymore
  - Detection based on:
    - Digital watermarks embedded into data (e.g. png-images), provided images are accessible by a web crawler (watermarking checker)
    - Visual watermarks embedded into data
Core Design: Data Leakage Detection

AB contains:
- $\text{Enc} [\text{Data}(D)] = \{\text{Enc}_{k_1}(d_1), \ldots, \text{Enc}_{k_n}(d_n)\}$
- Access Control Policies ($P$) = $\{p_1, \ldots, p_k\}$

- Service X is authorized to read $d_1$ from AB
- Service X may leak decrypted $d_1$ or the entire AB to Y
Core Design: Data Leakage Detection

- When service tries to decrypt AB data, CM is notified about that: “Service Y tries to decrypt $d_1$ arrived from X“
- If CM is unreachable, decryption terminates
- CM checks against centralized DB of policies: whether $d_1$ is supposed to be at Y. If NO then:
  - Blacklist X, Y
  - Reduce their trust level
  - Mark data $d_1$ as compromised and notify services about it
  - Raise the level of $d_1$ classification
Anti-fragility

- After leakage is detected, make system stronger against similar attacks
  - Separate compromised role into two: suspicious_role and benign_role
  - Send new certificates to all benign users for benign role
  - Create new Active Bundle with new policies, restricting access to suspicious_role (e.g. to all doctors from the same hospital with a malicious one)
  - Increase sensitivity level for leaked data items, i.e. for diagnosis
  - Disable “Save As” functionality or exclude highly sensitive data from what can be stored locally
Layered Approach: Don't give all the data to the requester at once
  - First give part of data (incomplete, less sensitive)
  - Watch how it is used and monitor trust level of using service
  - If trust level is sufficient – give next portion of data

Raise the level of data classification to prevent leakage repetition

Intentional leakage to create uncertainty and lower data value

Monitor network messages
  - Check whether they contain e.g. credit card number that satisfies specific pattern and can be validated using regular expressions [14]
Encrypted Search over Encrypted Data (suggested by Dr. Leon Li, NGC)

Example:

```
select prescription from Hospital_IS where diagnosis = "Insomnia";
```
Encrypted Search over AB Database

- Collection agent gathers intelligence feeds (ABs)
- AB contains extra-attribute used for indexing
- CryptDB is a proxy to a database server
  - Stores encrypted data (keywords, abstract of AB) and provides SQL query capability over encrypted data
  - Never releases decryption key to a database
  - When compromised, only ciphertext is revealed and data leakage is limited to data for currently logged in users
- Subscription API provides methods for authorized access to data
  - Phase 1: filter out relevant ABs (e.g. top-20)
  - Phase 2: execute data request to relevant ABs only [6]
- Use case 1:
  - Get prescription for patients diagnosed with “Insomnia”
    `select prescription from Hospital_IS where diagnosis = "Insomnia";`

!!! Note: due to vulnerabilities, recently discovered in CryptDB, it is recommended to use Microsoft SQL Server 2016 instead
Use case 2: law enforcement needs personal data of drivers who exceeded speed limit of 65 mph and went above 76 mph

**Initial Query:** SELECT ID FROM IndexDB WHERE SPEED > 76

**Converted query:** SELECT c1 FROM Alias1

WHERE ESRCH (Enc(Speed), Enc(76));

**Second phase query:** http get request for driver’s license number from VRs with relevant IDs from previous query
Encrypted Search over Encrypted Records

**Index Database**

<table>
<thead>
<tr>
<th>ID</th>
<th>Speed</th>
<th>Model</th>
<th>Timestamp</th>
</tr>
</thead>
<tbody>
<tr>
<td>Enc(001)</td>
<td>Enc(65)</td>
<td>Enc(Toyota)</td>
<td>02/18/2018 15:28</td>
</tr>
<tr>
<td>Enc(002)</td>
<td>Enc(66)</td>
<td>Enc(Ford)</td>
<td>02/18/2018 15:29</td>
</tr>
<tr>
<td>Enc(003)</td>
<td>Enc(67)</td>
<td>Enc(Mercedes)</td>
<td>02/18/2018 15:31</td>
</tr>
<tr>
<td>Enc(004)</td>
<td>Enc(68)</td>
<td>Enc(Mitsubishi)</td>
<td>02/18/2018 15:44</td>
</tr>
<tr>
<td>Enc(1000)</td>
<td>Enc(84)</td>
<td>Enc(Chevrolet)</td>
<td>02/18/2018 23:59</td>
</tr>
</tbody>
</table>

**Use case 3:** ITS needs to figure out traffic pattern during rush hour. Speed between 55 and 65 => no traffic

**Initial Query:** `select ID from IndexDB WHERE speed between 55 and 65`

**Converted query:** `SELECT c1 FROM Alias1 WHERE ERANGE (Enc(Speed), Enc(55), Enc(65));`

**Second phase query:** `http get request for vehicle’s license plate number from VRs with relevant IDs from previous query`
<table>
<thead>
<tr>
<th>Crypto System</th>
<th>Supported operations</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pallier (AHE)</td>
<td>+, SUM</td>
<td>Count sum of salaries</td>
</tr>
<tr>
<td>El-Gamal (MHE)</td>
<td>+, SUM, *</td>
<td>Count salary which is multiplication: hourly_wage * hours</td>
</tr>
<tr>
<td>OPE (Order-Preserving Encryption)</td>
<td>&gt;, &lt;, MIN, MAX, ...</td>
<td>Select patient from EHR_DB where AGE in between 25 and 35</td>
</tr>
<tr>
<td>SWP (SRCH)</td>
<td>Substring searches (LIKE in SQL queries)</td>
<td>Select prescription from EHR_DB where diagnosis LIKE %insomnia%</td>
</tr>
<tr>
<td>DET (deterministic)</td>
<td>Exact searches</td>
<td>Select patient from EHR_DB where Name = 'John Doe'</td>
</tr>
</tbody>
</table>
Performance overhead of Active Bundle with detection of browser's crypto capabilities on / off
Evaluation

Performance overhead of Active Bundle, hosted by Google Cloud
Secure Data Exchange in “WAXEDPRUNE” is based on [3]:

- Access control policies [16]
- Trust level of a subject (service, user)
- Context (e.g. emergency vs. normal)
- Security level of client’s browser (crypto capabilities) [12], [13]
- Authentication method (password-based, fingerprint, etc)
- Source network (secure intranet vs. unknown network)
- Type of client’s device: desktop vs. mobile (detected by Authentication Server)
- **Assumption: hardware and OS are trusted**
  - To relax these assumptions, Intel SGX trusted platform might be used
- Data is extracted from Active Bundle at a server side and send to client via https
  - Data confidentiality is preserved
- Multiple types of Data Leakages are prevented/detected by using:
  - Active Bundles
  - Digital watermarks, embedded into data. Watermarks are checked by web crawlers,
  - Visual watermarks
Conclusions

- CryptDB never releases decryption key to a database
  - provides database privacy
  - protects database from curious or malicious cloud administrators

- CryptDB weak points:
  - OPE is not secure in terms of revealing the order [19]
  - Does not support queries having \( a + b\times c \)
  - Does not support SQL queries with “LIKE”

**Solution:** use Fully Homomorphic Encryption (FHE)
  - It is 9x slower than CryptDB [9]
  - Pallier and El-Gamal don’t reveal order
Contributions

WAXEDPRUNE contributes to Data Confidentiality and Integrity

• Dissemination does not require data owner’s availability
• TTP-independent for recipient’s key generation
• Trust level of subjects is constantly recalculated
• On-the-fly key generation
• Supports data updates for multiple subjects
• Agnostic to policy language and evaluation engine
• Tamper-resistance: data and policies integrity is provided
Contributions

WAXEDPRUNE contributes to Data Confidentiality and Integrity

- Supports encrypted search over database of ABs
- Provides prevention/detection of multiple types of data leakages, made by malicious authorized insiders, and leakage damage assessment
- Captures data provenance for use in leakage measure and forensics
- Compatible with industry-standard SOA/cloud frameworks
  - RESTful services
  - X.509 certificates
References