Mylar

Building Web Applications on Top of Encrypted Data
Problem

- Web applications use servers to store and process confidential information.
  - Anyone who gains access to the server can obtain all of the data stored there.
Solution

Mylar assumes malicious or compromised server operator.

1. Mylar allows users to share keys and data securely in the presence of an active adversary (man in the middle attack or a malicious administrator actively tampering with the data sent to the client)
2. Mylar allows the server to perform keyword search over encrypted documents
3. Mylar ensures that client-side application code is authentic, even if the server is malicious.
Background
Mylar’s model

* Assumes that site owner is non-malicious.
Mylar’s model

* Assumes that site owner is non-malicious.
Question

- What does this design remind you of?
Question

- What does this design remind you of?

Here’s a hint...
Architecture

- **Browser extension**: Verify that the code of application has not been tampered with.
- **Client-side library**: Intercepts data sent to and from the server, and encrypts or decrypts that data.
- **Server-side library**: Performs computation over encrypted data at the server.
- **IDP**: Verify that a given public key belongs to a particular username.

*Mylar assumes that IDP correctly verifies a users identity*
Mylar’s threat model

1. Authenticate client-side application code
2. Data Sharing (Privately)
3. Keyword Search
#1 - Authenticate client side code integrity

1. Authenticate client-side application code
#1 - Authenticate client side code integrity

Source: https://www.google.com/search?q=mylar+ensuring+code+integrity&source=lnms&tbm=isch&sa=X&ved=2ahUKEwjciIH449TsAhUhneAKHT0DAiIQ_AUoAnoECAwQBA&biw=1527&bih=883#imgrc=2j1XzKe_ydvilM
#2 - Data Sharing

Client Browser

Encryption/decryption layer

Application server

Web Application

DB

2. Data Sharing (Privately)
#2 - Data Sharing

- **Client Browser 1**: Encryption/decryption layer
- **Client Browser 2**: Encryption/decryption layer
- **Application server**: Web Application
- **DB**: Database
Question

- Alice wants to send messages to Bob privately.
  - How does Mylar's client create a user?
  - How does it share documents?
Question

- Alice wants to send messages to Bob privately.
  - How does Mylar's create user?
  - Share document?

1. `create_user(uname, password, auth_princ)`
   - `auth_princ` can be either static principal or IDP
   - `auth_princ` helps generate certificate

2. a) Alice generates “Shared Document” pub/priv key pair
   b) Creates wrapped key $E(Priv_{Shared\ Doc}, Pub_{Alice})$
#2 - Data Sharing

user Alice

Has access to ->

chat room ‘party’
#2 - Data Sharing

- Bob’s principal is granted access to the chat room principal.
- Mylars client uses the public key of Bob to encrypt the document.
- Both Alice and Bob have access to the principal for “party”.
- Arrows: certificate chains to attest the mapping between principal name and public key.

![Diagram showing access to chat room principal](image)
#2 - Data Sharing

Messages are encrypted with the key for the room’s principal
#2 - Data Sharing - Key chaining

- Private key of ‘party’ is encrypted separately under the public key of Alice and Bob
- The same goes for the ‘work chat’ between Bob and Boss
- These keys are then ‘wrapped’ and stored on the server
#3 - Keyword Search
Question

- If a user wants to search for a word in a set of documents on the server, they are each encrypted with a different key. In terms of search, computation over one key at a time has serious limitations.

How does Mylar tackle this?
#3 - Keyword Search

- Only need to provide a single search token
  - The server, in turn, returns each encrypted document that contains the user’s keyword, as long as the user has access to that document’s key
- Use delta to adjust one token to another.
- Enable the server to compute token by itself.
#3 - Keyword Search

**Client-side operations:**

**procedure** `KEYGEN()`

- Generate a fresh key
  
  \[
  \text{key} \leftarrow \text{random value from } \mathbb{Z}_p
  \]

  **return** `key`

**procedure** `ENC(key, word)`

- `r` \leftarrow \text{random value from } \mathbb{G}_T
- `c` \leftarrow \langle r, H_2(r, e(H(\text{word}), g)^{\text{key}}) \rangle

  **return** `c`

**procedure** `TOKEN(key, word)`

- \( H(\text{word})^{\text{key}} \) \text{ in } \mathbb{G}_1

  **return** `tk`

**procedure** `DELTA(key_1, key_2)`

- Allow adjusting search token from `key_1` to `key_2`
  
  \[
  \Delta_{\text{key_1} \rightarrow \text{key_2}} \leftarrow g^{\text{key_2}/\text{key_1}} \text{ in } \mathbb{G}_2
  \]

  **return** `\Delta_{\text{key_1} \rightarrow \text{key_2}}`

**Server-side operations:**

**procedure** `ADJUST(tk, \Delta_{k_1 \rightarrow k_2})`

- Adjust search token `tk` from `k_1` to `k_2`
  
  \[
  \text{atk} \leftarrow e(tk, \Delta_{k_1 \rightarrow k_2}) \text{ in } \mathbb{G}_T
  \]

  **return** `atk`

**procedure** `MATCH(atk, c = \langle r, h \rangle)`

- Return whether `c` and `atk` refer to same word
  
  \[
  h' \leftarrow H_2(r, \text{atk})
  \]

  **return** `h' \stackrel{?}{=} h`
Limitations

- **4x Space Overhead for kChat**
  - Principal graphs (storing certificates and wrapped keys),
  - Symmetric key encryption
  - Searchable encryption

<table>
<thead>
<tr>
<th>Application</th>
<th>Operation for latency</th>
<th>Latency w/o Mylar</th>
<th>Latency with Mylar</th>
<th>Throughput w/o Mylar</th>
<th>Throughput with Mylar</th>
<th>Throughput units</th>
</tr>
</thead>
<tbody>
<tr>
<td>submit</td>
<td>send and read a submission</td>
<td>65 msec</td>
<td>606 msec</td>
<td>723</td>
<td>394</td>
<td>submissions/min</td>
</tr>
<tr>
<td>submit w/o search</td>
<td></td>
<td>70 msec</td>
<td></td>
<td></td>
<td>595</td>
<td></td>
</tr>
<tr>
<td>endometriosis</td>
<td>fill in/read survey</td>
<td>1516 msec</td>
<td>1582 msec</td>
<td>6993</td>
<td>6130</td>
<td>field updates/min</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Application</th>
<th>LoC before</th>
<th>LoC added for Mylar</th>
<th>Number and types of fields secured</th>
<th>Existed before?</th>
<th>Keyword search on</th>
</tr>
</thead>
<tbody>
<tr>
<td>kChat [23]</td>
<td>793</td>
<td>45</td>
<td>1 field: chat messages</td>
<td>Yes</td>
<td>messages</td>
</tr>
<tr>
<td>endometriosis</td>
<td>3659</td>
<td>28</td>
<td>tens of medical fields: mood, pain, surgery, ...</td>
<td>Yes</td>
<td>N/A</td>
</tr>
<tr>
<td>submit</td>
<td>8410</td>
<td>40</td>
<td>3 fields: grades, homework, feedback</td>
<td>Yes</td>
<td>homework</td>
</tr>
<tr>
<td>photo sharing</td>
<td>610</td>
<td>32</td>
<td>5 fields: photos, thumbnails, captions, ...</td>
<td>Yes</td>
<td>N/A</td>
</tr>
<tr>
<td>forum</td>
<td>912</td>
<td>39</td>
<td>9 fields: posts body, title, creator, user info, ...</td>
<td>No</td>
<td>posts</td>
</tr>
<tr>
<td>calendar</td>
<td>798</td>
<td>30</td>
<td>8 fields: event body, title, date, user info, ...</td>
<td>No</td>
<td>events</td>
</tr>
<tr>
<td>WebAthena [8]</td>
<td>4800</td>
<td>0</td>
<td>N/A: used for code authentication only</td>
<td>Yes</td>
<td>N/A</td>
</tr>
</tbody>
</table>

![Graphs showing performance metrics](image)
Conclusion

Mylar supports

- Keywords search over documents encrypted with different keys
- In the presence of an active adversary, share keys and encrypted data safely
- Verify Client-side application code
- Few changes to an application, and modest performance overheads
- Cannot guarantee data freshness, or correctness of query results.
Discussion

- Thoughts?
- What are some challenges to this model?
- Is this model applicable to large scale applications?
- How is Mylar different from CryptDB?