Mylar

Building web applications on top of encrypted data

Raluca Ada Popa, Emily Stark, Jonas Helfer, Steven Valdez, Nickolai Zeldovich, M. Frans Kaashoek, and Hari Balakrishnan

MIT CSAIL and Meteor Inc.
Problem

Confidential data leaks from web servers
Attackers get full server access

users

web server

Secret
Secret

hackers

cloud employees

government

LivingSocial Hacked — More Than 50 Million Customer Names, Emails, Birthdates and Encrypted Passwords Accessed (Internal Memo)

Why You Shouldn't Trust Facebook with Your Data: An Employee's Revelations
Mylar

A web framework that protects confidentiality against fully compromised servers
Servers store data encrypted

Plaintext data exists only in browsers
Related work

• File systems: CFS, NCryptfs, SiRiUS, Plutus
• Encrypted databases: CryptDB, Monomi
• Browser encryption: Christodorescu’08, Cryptocat

Far from sufficient for real web apps
Challenges

• Active adversaries (e.g., corrupt webpage)

• Enabling functionality with encryption:
  – data sharing
  – computation
Mylar

• **Active adversaries** (e.g., corrupt webpage)
• **Enabling functionality with encryption:**
  – data sharing
  – computation

webpage code verification   principal graph & certification
client-side web framework   new encryption scheme:
                           multi-key search
Example: Chat application

TODO:

- Server cannot see messages
- Users share chat rooms securely
- Format messages, generate HTML page
- Search
How to organize a web application framework for encryption?
Start: common web framework

e.g., Django, Ruby on Rails
Add encryption

- server’s computation is restricted by encryption
- easy to tamper with webpage
Client-side web framework

e.g., AJAX programming, Meteor

Data and code separate

Generate webpage at client, compute in browser
Mylar

Certify code (trusted developer)

Intercept and encrypt/decrypt data
Chat application

- Server cannot see messages
- Format messages, generate html page
- Users share chat rooms securely
- Search
Data sharing

Developer specifies access control via the principal graph

In Alice’s browser:

function create_chat(chattitle):
  chat_princ = princ_create(chattitle, princ_current());

function invite_user(username):
  chat_princ.add_access(
    princ_lookup(username));

function send_message(msg):
  Messages.insert(
    {"message": chat_princ});

Server database:

<table>
<thead>
<tr>
<th>message</th>
<th>chat</th>
<th>chatprinc</th>
</tr>
</thead>
<tbody>
<tr>
<td>‘SSN is 32..’</td>
<td>WORK</td>
<td>WORK princ</td>
</tr>
</tbody>
</table>
Enforce access with key chains

Server stores:

- WORK
  - ‘SSN is 32..’
- PARTY
  - ‘7pm on Saturday’
  - ‘at my place’

Alice

Bob

Eve
Get access to shared data

Server stores:

WORK
‘SSN is 32..’

PARTY
‘7pm on Saturday’
‘at my place’

Bob

‘SSN is 32..’

Eve

‘SSN is 32..’
Problem: attacker gives incorrect key

Encrypt message for **WORK**

Want key

Receive key

Server stores:

- **WORK**
  - ‘SSN is 32..’

- **PARTY**
  - ‘7pm on Saturday’
  - ‘at my place’

Eve

Has access to !!!

Bob
Solution: Certification graph

- $\text{sign}_{\text{IDP}}(\text{Alice, Work})$
- $\text{sign}_{\text{Alice}}(\text{WORK, SSN is 32..})$

How does Bob’s browser know
1. that it needs to check a signature from Alice?
2. Alice’s PK?  IDP: invoked once per user account creation
Choosing the certification path

1. Principals have human meaningful names
2. Developer displays entire path
3. User chooses path

No other change to user experience!
Chat application

- Server cannot see messages
- Format messages, generate html page
- Users share chat rooms securely

Search
Challenge: multi-key

Server:

Bob

WORK
‘SSN is 32..’

PARTY
‘7pm on Saturday’
‘at my place’
Strawman: use single-key search scheme

[Kamara et al.’12]

Server:

\[\text{WORK}
\begin{align*}
\text{‘SSN is 32..’}
\end{align*}
\]

\[\text{PARTY}
\begin{align*}
\text{‘7pm on Saturday’} \\
\text{‘at my place’}
\end{align*}
\]

\[\text{Match!}\]

\[\text{Slow: pay overhead for each key}\]
New cryptosystem: multi-key search

Server adjusts encryption from one key to another

Based on elliptic curves

**API:**
- Setup
- Keygen
- Encrypt
- Token
- Delta
- Adjust
- Match
Delta

In Bob’s browser:

WORK:
\[ \text{Delta}() \to \begin{array}{c} \text{Key 1} \\ \text{Key 2} \end{array} \]

PARTY:
\[ \text{Delta}() \to \begin{array}{c} \text{Key 3} \\ \text{Key 4} \end{array} \]
Chat application

- Server cannot see messages
- Format messages, generate html page
- Users share chat rooms securely
- Search
Confidentiality guarantees

Protects user A’s data confidentiality against
• full server compromise
• compromise of any user machine, except for users with legitimate access to user A’s data
assuming
• developer’s client-side code does not leak data

Does not protect against side channels or access patterns, and does not hide metadata
Implementation

• On top of Meteor, but design is not limited to Meteor

• 9000 LoC: Javascript and C++
Evaluation

• How much developer effort does porting apps require?

• What is the performance overhead?
### Applications

<table>
<thead>
<tr>
<th>Applications</th>
<th>Fields secured</th>
<th>LoC added</th>
<th>LoC total</th>
<th>Existed before</th>
</tr>
</thead>
<tbody>
<tr>
<td>kChat</td>
<td>chat messages</td>
<td>45</td>
<td>793</td>
<td>Yes</td>
</tr>
<tr>
<td>endometriosis</td>
<td>medical fields</td>
<td>28</td>
<td>3659</td>
<td>Yes</td>
</tr>
<tr>
<td>class submit</td>
<td>grades, homework, feedback</td>
<td>40</td>
<td>8410</td>
<td>Yes</td>
</tr>
<tr>
<td>photo sharing</td>
<td>photos, thumbnails, ..</td>
<td>32</td>
<td>610</td>
<td>No</td>
</tr>
<tr>
<td>forum</td>
<td>post body, title, ..</td>
<td>39</td>
<td>912</td>
<td>No</td>
</tr>
<tr>
<td>calendar</td>
<td>event body, title, ...</td>
<td>30</td>
<td>798</td>
<td>No</td>
</tr>
</tbody>
</table>

≈36 LoC
Experimental setup

Client

- Eight 10-core Intel Xeon E7-8870
- 2.4 GHz
- 256 GB of RAM

5 Mbit/s, 20 msec round-trip

Web server

- Intel Xeon 2.8 GHz
- 4 GB of RAM
Figure 9: End-to-end latency of four operations in kChat. Transmit includes the time from when one user sends a message to when another user receives it. Figure 9 shows the end-to-end latency Mylar introduces for four main operations in kChat: transmitting a message, joining a room, searching for a word in all rooms, and inviting a user to a room. For message transmission, we measured the time from the sender clicking “send” until the message renders in the recipient’s browser. This is the most frequent operation in kChat, and Mylar adds only 50 msec of latency to it. This difference is mostly due to searchable encryption, which takes 43 msec. The highest overhead is for inviting a user, due to principal operations: looking up and verifying a user principal (218 msec) and wrapping the key (167 msec). Overall, we believe the resulting latency is acceptable for many applications, and subjectively the application still feels responsive.

We also measured the latency of initially loading a page. The original kChat application loads in 291 msec. The Mylar version of kChat, without the code verification extension, loads in 356 msec, owing to Mylar’s additional code. Enabling the code verification extension increases the load time to 1109 msec, owing to slow signature verification in the Javascript-based extension. Using native code for signature verification, as we did for bilinear pairings, would reduce this overhead. Note that users experience the page load latency only when first navigating to the application; subsequent clicks are handled by the application without reloading the page.

We also measured the end-to-end latency of the most common operations in the endometriosis application (completing a medical survey and reading such a survey), and the submit application (a student uploading an assignment, and a staff member reading such a submission); the results are shown in Figure 11. For the submit application, we used real data from 122 students who used this application during the fall of 2013 in MIT’s 6.858 class. Submit’s latency is higher than that of other applications because the amount of data (student submissions) is larger, so encryption with search takes longer.

Figure 10: Server throughput for kChat. For comparison, we also show the latency of submit when search is turned off. The search encryption can happen asynchronously so the user does not have to wait for it.

To measure Mylar’s impact on server throughput, we used kChat, and we set up many pairs of browsers—a sender and a receiver—where the sender continuously sends new messages. Receivers count the total number of messages received during a fixed interval. Figure 10 shows the results, as a function of the total number of clients (each pair of browsers counts as 2 clients). Mylar decreases the maximum server throughput by 17%. Since the server does not perform any cryptographic operations, Mylar’s overhead is due to the increase in message size caused by encryption, and the encrypted search index that is added to every message to make it searchable.

Figure 11 also shows the server throughput of the endometriosis and class submit application when clients perform representative operations.

Search. To evaluate the importance of Mylar’s multi-key search, we compare it to two alternative approaches for secure search. The first alternative is single-key server-side search, in which the client generates a token for every key by directly computing the adjusted token from our multi-key search. This alternative is similar to prior work on encrypted keyword search. In this case, the client looks up the principal for every room, computes a token for each, and the server uses one token per room. The second alternative is to perform the search entirely at the client, by downloading all messages. In this case, the client still needs to look up the principal for each room so that it can decrypt the data.

Figure 12 shows the time taken to search for a word in kChat for a fixed number of total messages spread over a varying number of rooms, using multi-key search and the two alternatives described above. We can see that multi-key search is much faster than either of the two alternatives, even with a small number of rooms. The performance of the two alternatives is dominated by...
Mylar

• A web platform that protects confidentiality against full server compromise
  – Secures real applications with few LoC
  – Modest overhead

webpage code verification principal graph & certification
new encryption scheme: multi-key search

http://css.csail.mit.edu/mylar/

Demo!