Building web applications on top of encrypted data using Mylar

By: Titako Takapu
Problem

- How can I build a web application that securely stores my data without losing functionality or efficiency?
- Is there something out there that can keep these three principles in mind?
MYLAR
What is Mylar?

- Mylar is used to protect confidential data from adversaries
- Assumes attackers have *full access* to servers
- Mylar encrypts/decrypts in the user’s browser (client)
- Since it is built on Meteor, its static html code and data are separate
Decrypted data exists only in users’ browsers
Why use Mylar?

- Mylar allows for different keys to be used, which enables multiple parties access to encrypted data i.e. Chat service

- Mylar allows for computation over encrypted data with multiple keys i.e. Encrypted keyword search
Simple Overview of Mylar

- Verifies the application code running on the client
- Uses Identity Provider Service (IDP) to ensure identity of user
- The client side code then is able to encrypt/decrypt relevant data
Simple Overview

Figure 1: System overview. Shaded components have access only to encrypted data. Thick borders indicate components introduced by Mylar.
procedure PROCESSRESPONSE(url, cert, response)
  ▷ url is the requested URL
  ▷ cert is server’s X.509 certificate
  if cert contains attribute mylar_pubkey then
    pk ← cert.mylar_pubkey
    sig ← response.header["Mylar-Signature"]
    if not VERIFYSIG(pk, response, sig) then
      return ABORT
  if url contains parameter “mylar_hash=h” then
    if hash(response) ≠ h then return ABORT
  return PASS

Figure 5: Pseudo-code for Mylar’s code verification extension.
Sharing Data

- Access graph- uses key chains to distribute private keys of shared principals to users (principal is a public/private key pair) for application level access control entity

- Certification graph- uses certificate chains to attest the mapping between principal name and public key
Access graph example

- Key Chaining- Like CryptDB, Mylar stores wrapped keys on the server
Certificate graph example

- Adds another field like user name to ensure that the user is giving the key to the correct principal
Computing over Encrypted Data

- Must search through every word of the document if multiple
- There is a index version, that removes randomness without comprising security
  - needed to hide whether two words encrypted under the same key are equal
Multi Key Searching Algorithm

Client-side operations:

procedure KEYGEN() ▷ Generate a fresh key
    key ← random value from \( \mathbb{Z}_p \)
    return key

procedure ENC(key, word)
    \( r \leftarrow \) 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)
    ▷ Generate search token for matching \( \text{word} \)
    \( tk \leftarrow H(\text{word})^{\text{key}} \) in \( \mathbb{G}_1 \)
    return \( tk \)

procedure DELTA(key1, key2)
    ▷ Allow adjusting search token from \( key_1 \) to \( key_2 \)
    \( \Delta_{key_1 \rightarrow key_2} \leftarrow g^{key_2/key_1} \) in \( \mathbb{G}_2 \)
    return \( \Delta_{key_1 \rightarrow key_2} \)

Server-side operations:

procedure ADJUST(tk, \( \Delta_{k_1 \rightarrow k_2} \))
    ▷ Adjust search token \( tk \) from \( k_1 \) to \( k_2 \)
    \( atk \leftarrow e(tk, \Delta_{k_1 \rightarrow k_2}) \) in \( \mathbb{G}_T \)
    return \( atk \)

procedure MATCH(atk, \( c = \langle r, h \rangle \))
    ▷ Return whether \( c \) and \( atk \) refer to same \( \text{word} \)
    \( h' \leftarrow H_2(r, atk) \)
    return \( h' \approx h \)