VSRI Private Document Exchange

Overview

The risk areas for secure document exchange include...

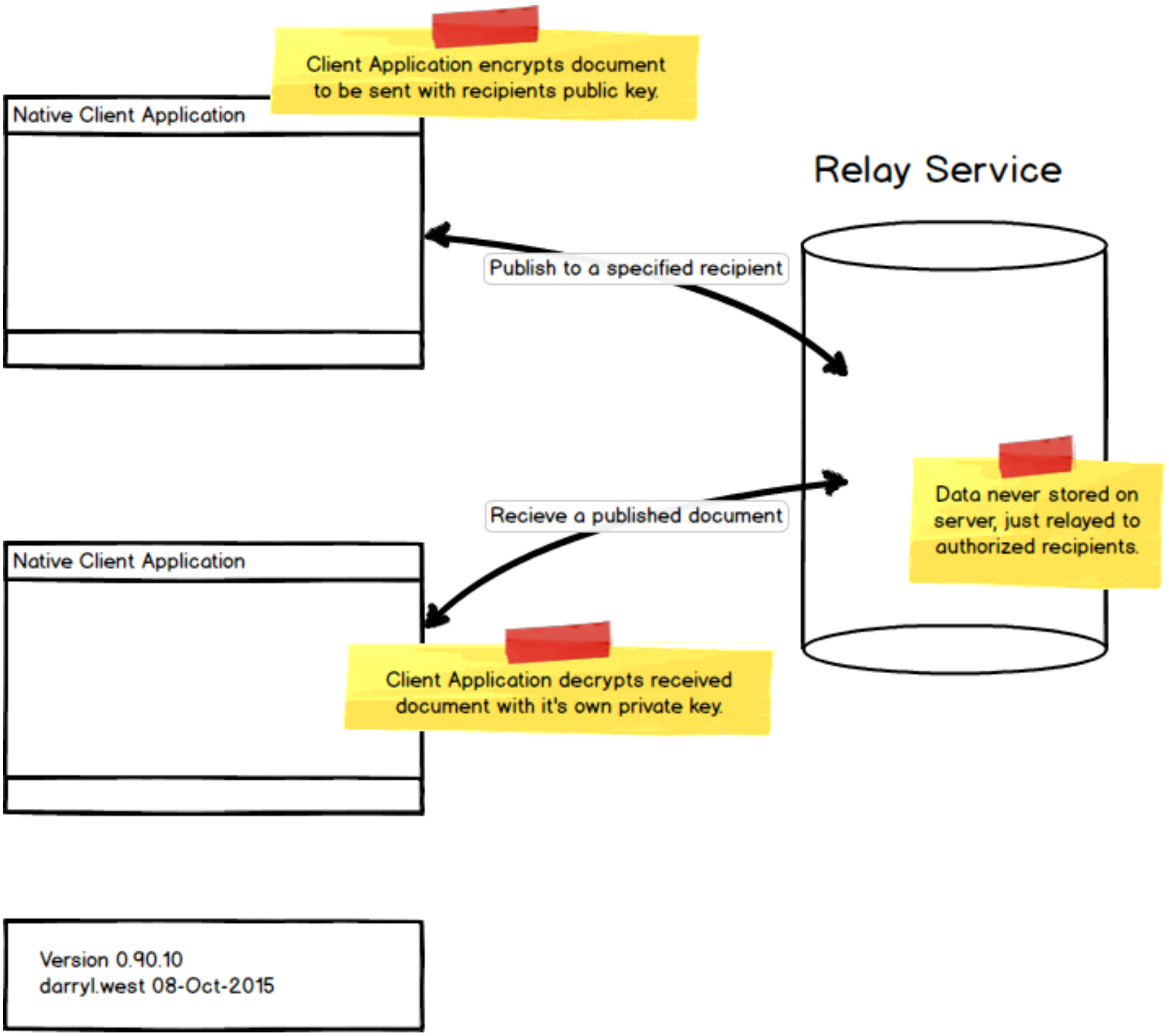
- data transmission, man-in-the-middle attacks
- data at rest being compromised by a hacker
- authentication breech caused by careless users, weak passwords, etc
- authentication breech caused by stolen device

Our solution addresses these risks by creating a secure client-to-client environment where only the recipient has the ability to decrypt the received document. Each user would have their own private copy of the client application, compiled just for them.

Combining hmac document signatures with secure sockets solves the man-in-the-middle risk. Using machine keys to encrypt sensitive documents solves the careless user/weak password risk.

Our relay service acts as temporary storage until the recipient requests a document published specifically for them. This solves the data-at-rest risk.

Client applications include additional back-channel methods of authenticating users via SMS, email, or other means. This additional authentication solves the stolen device risk.


Client Application encrypts document to be sent with recipients public key.
Native Client Application
Relay Service
Publish to a specified recipient
Recieve a published document
Data never stored on server, just relayed to authorized recipients.
Native Client Application
Client Application decrypts received document with it's own private key.
Version 0.90.10
darryl.west 08-Oct-2015

Document Exchange Technology

Client Application

The client application has the ability to send out it's public key but never transmits it's private key. Client applications are written in nw.js (NodeWebkit) with low-level exchanges done in go/c++ or some other compiled language. Target environments include Windows, MacOSX and Linux. Additional phase 2 versions could support iOS/iPad/iPhone.

- client applications compiled for Windows, Mac and Linux

- client application can act as a document producer or consumer
- each user would receive a unique version with it's own ID and set of keys
- as a recipient, read only documents would not be stored
- read/write documents would enable changes to the document to be transmitted back to the publisher
- admin documents would provide the ability to store the document on the recipient's device (they then potentially become a content producer)
- all exchanges would use the recipient's public key to encrypt data by the producer
- all exchanges travel through the relay server as a temporary holding area until the recipient is ready to receive

Relay Service

The relay service operates over secure websockets, only handles encrypted anonymous data transfers, and never persists data. This allows hosting in any environment that supports high data throughput, e.g., relatively low cost Amazon services.

The main features/functions are...

- handle document transfer requests from publishers by buffering data (nothing permanently saved to disk)
- handle recipient requests to pull a specified document
- receive data from publishers via secure websockets and hold in memory
- transfer data to recipients via secure websockets
- send out back-channel authentication request
- process back-channel authentications

The relay service would be written in go/java/c++ or other secure, compiled language. TLS1.2+ would be used for all transfers. Back channel communications would be delivered through SMS, email and/or custom service.

Summary

This solution leverages the relatively small user set (10K) to create an exchange environment that does not rely on user passwords. Back channel authentication provides extra protection and security. This level of security would be impossible to do inside a browser based application.

The relay system anonymously takes bits of encrypted data and holds it for transfer to the specified recipient who has the only capability to decrypt the data.

---

darryl.west@roundpeg.com version 0.90.11 10-Oct-2015