

CS2361: Blockchain and Cryptocurrencies

Project Milestone 1: M-Pin

Argha Chakrabarty Aarav Varshney

April 13, 2022

Introduction

We need authentication for primarily three reasons:

- authenticate the client to the server,
- authenticate the server to the client,
- and should result in a negotiated encryption key with which subsequent communications can be encrypted.

Until now we've been using Username/Password authentication for authenticating the client and the use of SSL/TLS protocols for authenticating the server. SSL even though now deprecated still had some good ideas but the Username/Password is extremely vulnerable to exploits and that's why there is a massive shift to Multi Factor Authentication (MFA).

The biggest exploit for username/password authentication is that the server stores either the hash of the password or the password itself in the database which if compromised can be used to gain access to the passwords.

The idea behind M-Pin is that each registered client is issued with a large cryptographic secret. They then prove to the server that they are in possession of this secret using a zero-knowledge proof. This removes the requirement for any information related to client secrets to be stored on the server.

Another crucial attribute of M-Pin is the use of third party authentication. Similar to how SSL uses a CA to verify the certificates, M-Pin uses Trusted Authority (TA) to store the secrets in contrast to Username/Password where the server performs regular operations as well as authentication.

Technical Details

- Pairing Based Cryptography: It is based on pairing functions that map pairs of points on an elliptic curve into a finite field.
- Identity Based Encryption: When communicating with someone using their public key, there is always a concern whether the public key belongs to intended party or some adversary. We see the usage of CAs to verify the legitimacy of the public key in PKIs. IBE proposed that the public key should be composed of some identifying information such as email address.
- Replay Attack: The attacker can capture the encrypted message sent to a trusted party and send it again at a later time. For instance, if an attacker captures a request for a financial txn, they can send the request again to make another txn.

The process for authentication is as follows:

1. Key Generation: The client computes point A on the elliptic curve using a ID_a which is then sent to the TA to get client secret key sA where s is a master secret stored in the TA.
 - (a) Notations:
 - i. ID_a : The client's email address.
 - ii. H_1 a hash function which maps to a point on G_1
 - iii. H_2 a hash function which maps to a point on G_2
 - iv. $e : G_1 \times G_2 \rightarrow Z_q$
 - v. sA : The client's secret key.
 - (b) Process:
 - i. Client: $A \leftarrow H_1(ID_a)$
 - ii. TA: Calculates sA
 - iii. Client: The client calculates the token $((s - \alpha)A)$ using his own PIN (α) and stores the token in local storage.
2. Time Permit: Added layer of security to the client's authentication. The client sends a time permit to the server which is also individually calculated by the server.
 - (a) Notations:
 - i. T_i : Today's date.
 - ii. T : The client's time permit.
 - (b) Process:

- i. Client: $T \leftarrow H_1(T_i||ID_a)$
 - ii. TA: Calculates sT
 - iii. Client: Stores sT in local storage.
3. Authentication: Communication between the client and the server takes place.
 - (a) Notations:
 - i. 'x' a random number in Z_q
 - ii. 'y' a random number in Z_q
 - (b) Process:
 - i. Client: Receives pin α from the user, picks x.
 - ii. Client: Computes $D \leftarrow A + T, U \leftarrow xD$.
 - iii. Client: Sends $\{ID_a, U\}$ to the server.
 - iv. Server: Sends y to the client.
 - v. Client: Computes $V \leftarrow -(x+y)((s-\alpha)A + \alpha A + sT)$ and sends V to the server.
 - vi. Server: Computes $D \leftarrow H_1(ID_a) + H_1(T_i||ID_a)$ and $g \leftarrow e(V, Q)e(U + yD, sQ)$.
 - vii. If $g = 1$ then the client is authenticated.

The entire process suffers from one flaw which is that it takes three passes between the client and the server which if the server also carries out user-name/password based authentication will find it difficult to change protocol. The paper titled "Milagro Multi-Factor Authentication" proposes e-M-Pin that uses a single pass between the client and the server.

The one pass authentication poses the problem of Replay Attack which is why we now also use current time: a) CCT: Current Client Time, b) SCT: Current Server Time and a single use arbitrary number, nonce.

The new process is only slightly different:

1. The client now also generates a nonce in Z_q along with x and stores the current time.
2. y is now computed as: $y \leftarrow H_q(ID_a||U||W||nonce||CCT)$.
3. and the client sends: $ID_a, U, W, V, nonce, CCT$ to the server.
4. The server first checks if the SCT - CCT falls within the expiration time (t) else fail the authentication. Secondly, verify if the nonce has been used already or not.
5. Finally, save the nonce and calculate y as $y \leftarrow H_q(ID_a||U||W||nonce||CCT)$.

Plan for the Project

1. Client Side: We are using Typescript, a superset of javascript for the client-side implementation of the language.
 - For the front-end app we are using react-ts.
 - Along with react, for the client side computation of the tokens and keys, `crypto-js` will be used.
 - For data fetching and mutation we will be using `axios`
2. Server Side:
3. TA:

Future Ideas / Plans for expansions

References

- [1] Masahiro Matsui et al. Milagro multi-factor authentication.
<https://www.ntt-review.jp/archive/ntttechnical.php?contents=ntr201612ra1.html>.
- [2] Michael Scott. M-pin: A multi-factor zero knowledge authentication protocol. <https://miracl.com/assets/pdf-downloads/mpin4.pdf>.