SplitSecure
A Distributed Credential Storage System

P. Kaundinya, D. Narayanan, Q. Nguyen, R. Mahajan
{pranavk, deepakn, qdnguyen, rohanm}@ mit.edu

12^{th} December, 2013

1 Abstract

Password databases of several high profile companies such as Sony, LinkedIn, and Adobe have been compromised in recent years. These security breaches affect millions of users and result in millions of dollars in damage along with a lot of negative publicity. Despite the obvious security hazards, companies often store passwords in clear text or hashed but unsalted form. As computing power becomes cheaper, even salted passwords are becoming vulnerable.

We designed a scalable distributed server-side password storage system with the objective of forcing adversaries to compromise multiple servers in order to obtain any sensitive information.

2 Related Work

Although distributed cryptography is a well established field, prior to 2012, there were no commercially available distributed password storage and authentication systems. With security breaches becoming more frequent and computation becoming cheaper, there is now a market for such a distributed password storage system.

Some key ideas in our design are inspired by the following prior work.

2.1 Distributed Credential Protection (DCP)

The first commercially available distributed password storage system was released by RSA in 2012 and is called Distributed Credential Protection (DCP) [1]. This scheme forces an adversary to compromise multiple servers in order to obtain any useful information. It stores the password using a combination of two servers - one server which stores the password XORed with a random number and the other which stores the random number itself. The system verifies the password without reconstructing it. The mechanics of this scheme are shown in **Figure 1**.

Scalability is a concern in this system because during authentication, the server needs to contact all databases which are involved in storing the user's password.

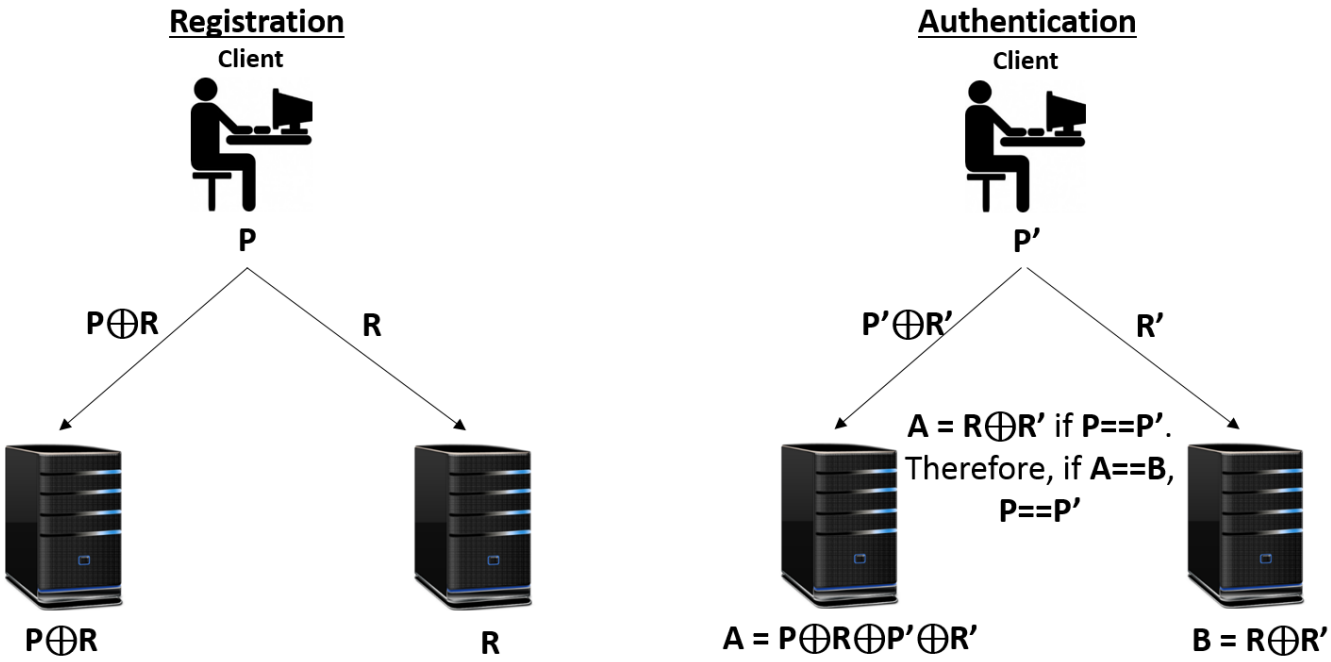

Figure 1: *RSA Distributed Credential Protection [1]: DCP stores each password using two servers - one server which stores the password XORed with a random number and the other which stores the random number itself. During registration, the client sends a password XORed with a random number to one server and the random number to the other server. During authentication, the client sends the password XORed with a new random number to the first server and the random number to the second. The system verifies the password without reconstructing it.*

2.2 Shamir's Secret Sharing Scheme

Shamir's Secret Sharing [2] is a k-out-of-n threshold secret sharing scheme in which a secret is split among a group of n entities. Each entity holds a unique share, and any k of these n entities are sufficient reconstruct the secret.

The core idea behind Shamir's Secret Sharing scheme is that a polynomial of degree $k-1$ can be uniquely determined by k points. If a secret is encoded as the constant term of a degree $k-1$ polynomial with randomly chosen coefficient and arbitrary points on the polynomial are chosen as shares of the secret, k shares are sufficient to reconstruct the polynomial and obtain the secret. The individual points by themselves contain no information about the polynomial whatsoever. Shamir's Secret Sharing scheme can be shown to be information theoretically secure.

3 Design

We designed our system with the following design goals:

- To never reconstruct user passwords or store complete passwords on any one machine

- To make the system more scalable than existing systems
- To minimize the computation load on the client

Our system consists of two main components:

- **Authentication server:** This is the main machine that the client communicates with while registering and authenticating.
- **Database servers:** These are the machines which actually store password information. Let the number of database servers containing a share of any given user's password be n, and the number of databases involved in authentication be k. Note that no server stores more than one share of a user's password.

We describe our design in more detail in the following sections.

3.1 Threat Model

The main aim of our system is to protect users' passwords in the event that one or more server-side machines are compromised. We assume that each server-side machine is well isolated so that attacking one server successfully doesn't make it any easier to attack the other servers.

Our design assumes that the network is safe. There are several existing solutions to deal with network vulnerabilities. The goal of our design is to protect against server attacks. All communication of sensitive data in our system is encrypted and we use certificates and signatures to verify authenticity. However, our design is still vulnerable to attacks where an adversary modifies packet routing or intercepts packets.

3.2 Registration

Consider the registration protocol shown in figure **Figure 2**. The following steps are involved in registration:

1. The client issues a registration request to the authentication server. This request contains the username of the user trying to register. The authentication server first checks if there already exists an entry for that particular username in the username-database servers table. If so, registration fails and the client is notified. Otherwise, the authentication server creates a new entry for the user in the username-database servers table. This table stores a list of n tuples for each user in which the first element in each tuple is the address of a database server and the second element is a randomly chosen challenge point.
2. The authentication server responds to the registration request with a list of n (database server, random challenge point) tuples that tell the client which databases it needs to talk to, along with the corresponding challenge points. The n database servers are chosen randomly from the pool of database servers available and recorded in the username-database server table. The authentication server also sends the client a digitally signed token containing the username. This token is used by the client to communicate with the database servers.

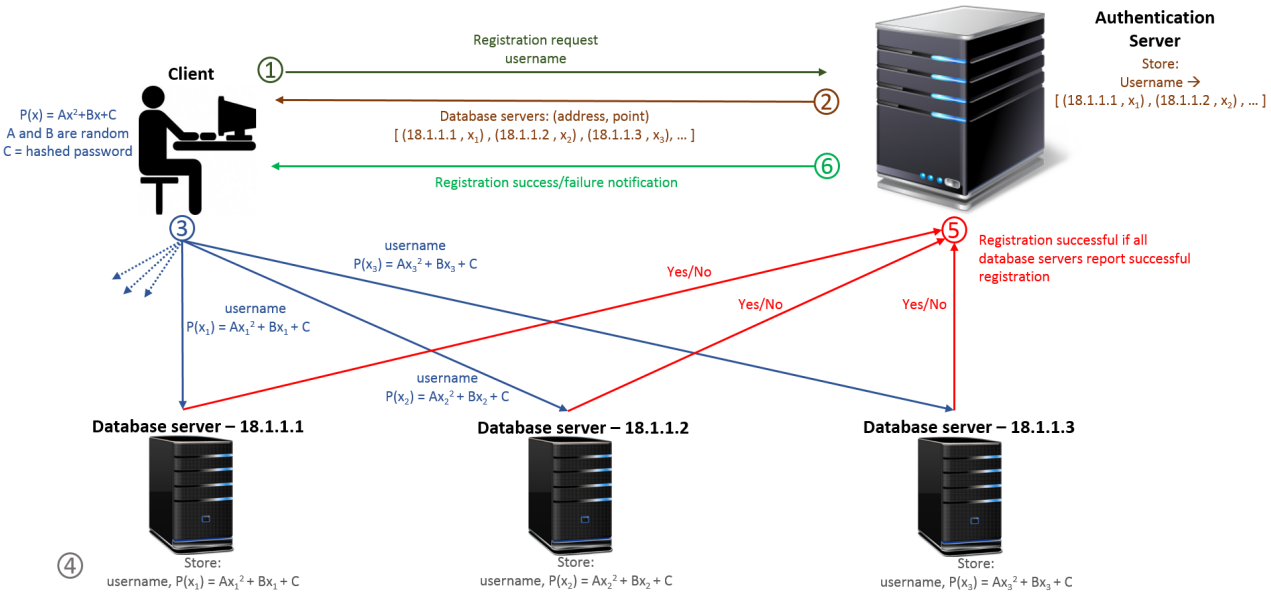

Figure 2: *The registration protocol in our distributed password storage system.*

3. The client then chooses random numbers A and B and hashes the entered password to obtain a number C. The client constructs the polynomial $P(x) = Ax^2 + Bx + C$, which is then evaluated at the challenge points obtained from the authentication server. These evaluated values (password shares) along with the client's username are then sent to the corresponding database servers. Communication between the client and authentication/database servers is protected by SSL to maintain secrecy and authenticity.

4. Each database server processes the request it gets from the client by storing the value obtained from the client (a password share) in a table mapping usernames to password shares. Before processing a request, the database server verifies the signed username in the token presented by the user.

5. Each database server notifies the authentication server of successful registration

6. The authentication server notifies the client that the registration was successful if it receives a successful registration signal from all the n database servers

3.3 Authentication

Consider the authentication protocol shown in figure **Figure 3**. The following steps are involved in authentication:

1. The client issues an authentication request to the authentication server. This request contains the username of the client. Using this username, the authentication server performs a lookup in the username-database servers table.

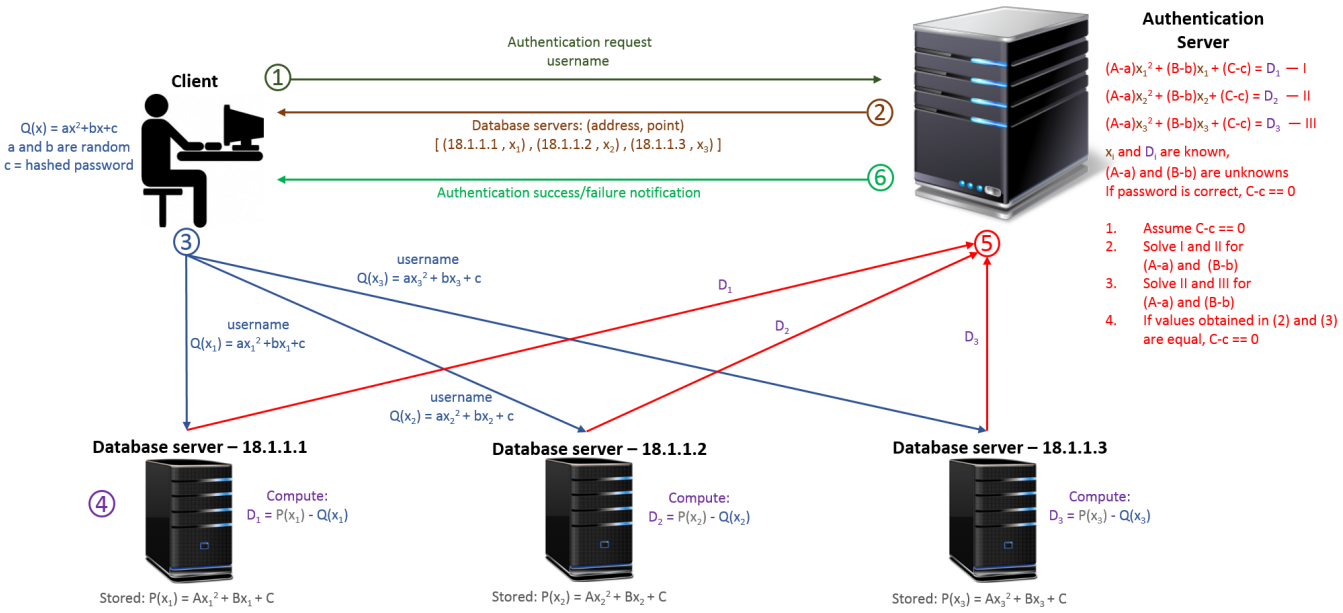

Figure 3: *The authentication protocol in our distributed password storage system for* $k = 3$.

2. The authentication server responds to the authentication request with a list of database servers that the client needs to talk to, together with the challenge points associated with those database servers. This list consists of k of the n database servers containing the user's password shares. To balance the load, the k database servers are chosen so that the more loaded database servers are less likely to be chosen. The authentication server also sends the client a digitally signed token containing the username. This token is used by the client to communicate with the database servers.

3. The client chooses random numbers a and b and hashes the password just entered by the user to obtain a number c (Note that these random numbers are different from those used during registration - A and B). The client constructs the polynomial $Q(x) = ax^2 + bx + c$ and then evaluates the polynomial Q at the challenge points obtained from the authentication server. These evaluated points are then sent (along with the client's username) to the corresponding database servers. As with registration, communication between the client and authentication/database servers is protected by SSL to maintain secrecy and authenticity.

4. Each database server processes the request it gets from the client by computing the difference D_k between the value stored in the database and the value obtained from the client. These differences are then sent to the authentication server, where the authentication process actually happens. Before processing a request, the database server verifies the signed username in the token presented by the user. The communication between the database servers and authentication servers is encrypted with a symmetric key that is renewed periodically.

5. The authentication server rewrites each difference as a polynomial with unknown coefficients.

It then assumes that the password is correct, thus obtaining a system of k equations and $k-1$ unknowns. By choosing different subsets of these equations, solving for the unknowns, and comparing the results, the authentication server can verify whether the password is correct. The steps involved in this computation are illustrated for $k = 3$ in figure **Figure 3**.

6. The authentication server notifies the client whether the authentication was successful or not.

4 Implementation

We implemented a simple prototype of our design to serve as a proof-of-concept. We implemented the authentication server and database servers as python HTTP servers. We developed a client with a simple user interface. We used Javascript to perform all the client-side computation.

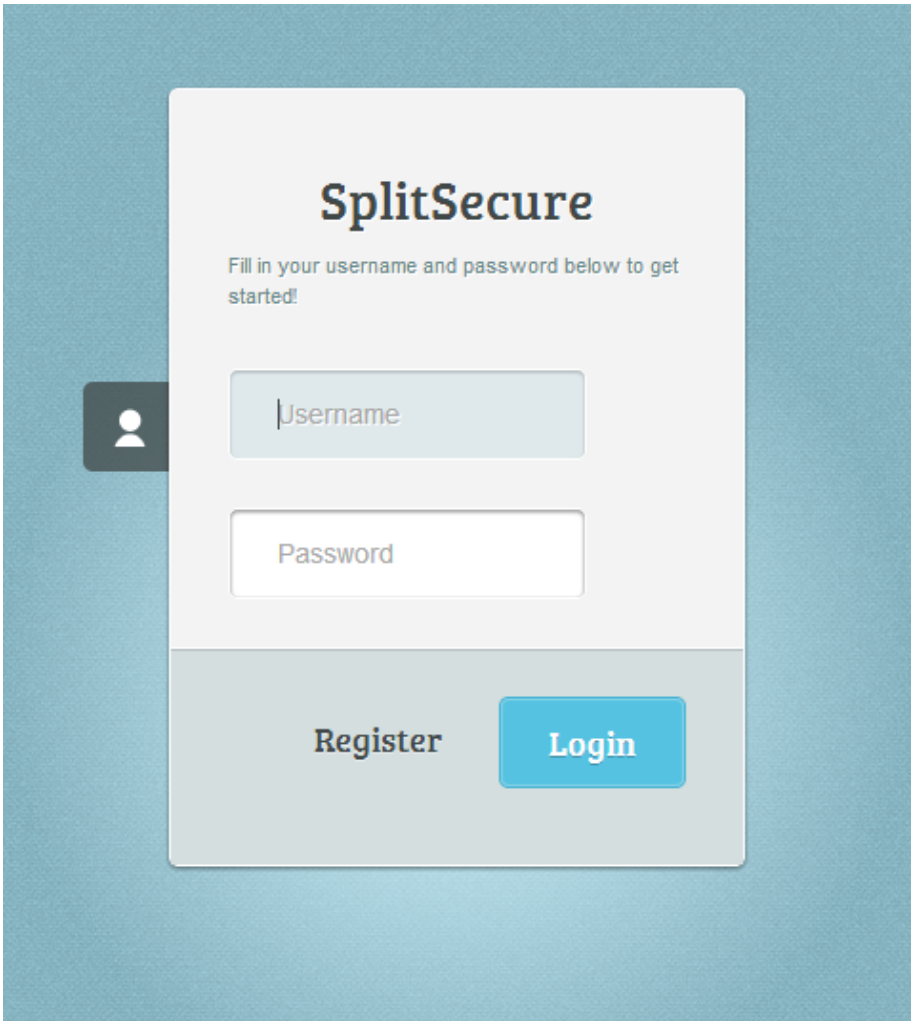

Figure 4: *User interface of our prototype implementation*

The security of our design is based on the assumption that successfully compromising one

database server doesn't make it any easier to compromise any of the other database servers. Therefore, successful implementation requires enforcing strong isolation between the database servers. One way of achieving this is to virtualize the database servers using different operating systems. For the purposes of our prototype, we assumed that such isolation methods were already in use.

5 Analysis

5.1 Security

- **Authentication Server Attacks:**
  Since no passwords are stored on the authentication server, the adversary doesn't obtain any passwords by compromising the authentication server. Further, the authentication servers only obtain the differences of shares from the database servers and the password isn't reconstructed during authentication. Even if an adversary listens to all communications with the client and aggregation servers, he doesn't obtain any sensitive information. In the worst case, an adversary could misdirect a client to a false database server, but such an database server would be unable to present a valid certificate to the client. Therefore the worst thing an adversary could do is to mount a denial-of-service attack.

- **Database Server Attacks:**
  Since each database only stores a single share of each password, the adversary doesn't obtain any sensitive information by attacking a few database servers. Compromising a database server would only give the attacker values of arbitrary polynomials evaluated at the challenge points. The challenge points can easily be obtained by simply querying the authentication server. However, a share stored in a database server provides no information about the polynomial (and therefore the password) even if the corresponding challenge point is known. An adversary needs to compromise k database servers in order to obtain any complete passwords since k points are necessary in order to determine a $k-1$ degree polynomial. The constant term of the polynomial (which is the hashed password) cannot be determined without reconstructing the entire polynomial.

5.2 Performance

Even though our scheme is more computationally expensive than a normal authentication scheme we don't believe that this excess computation is a major drawback. The excess computation is quite simple and there exist several efficient algorithms to perform this computation. Besides the standard cryptographic computations needed to make sure that data is sent securely over the network, the additional computation that our design introduces is two-fold - the client now has to evaluate a polynomial at n different points and the server needs to solve systems of linear equations. Evaluating the polynomial isn't very computationally intensive (especially for low degree polynomials) so the client isn't burdened by heavy computations. Solving the system of linear equations is more expensive, but this can be done reasonably efficiently using optimized matrix operations.

However, our system consumes significantly more network bandwidth than ordinary authentication schemes. To authenticate itself, the client needs to make k requests ($+1$ depending on the exact implementation) in our design instead of just a single request in normal authentication schemes. Since the data being sent in each request is small, for small values of k, we believe that this overhead is quite insignificant. The authentication server also needs to communicate with k database servers. For a large-scale system, this could be a significant overhead.

While there will certainly be an increase in the time taken for authentication, we don't expect this impact to significantly affect user experience. Even though we simulated some load, it was hard to get a quantitative measure of the impact on authentication time from our prototype. The fact that our scheme is scalable and allows load balancing makes it possible to implement the scheme in a way that minimizes the impact on authentication time. Further, we expect this system to be primarily used in cases where users' credentials are extremely valuable, so we believe that a slight increase in authentication time is acceptable.

5.3 Hardware Overhead

Our system requires powerful and robust authentication servers because they perform most of the computation. The authentication server also needs to store the database servers and challenge points corresponding to users so it needs to have a database associated with it. A complete large-scale implementation of our scheme would have multiple authentication servers.

Although database servers don't perform any intense computation, they handle requests directly from users and need to be capable of handling load. Our k-out-of-n scheme allows us for database fault-tolerance to be built into the implementation.

6 Conclusion

Using Shamir's k-out-of-n secret sharing scheme, we were able to design a password storage system that is more scalable than RSA's uted Credential Protection (DCP) at the cost of some extra computational overhead. Unlike in DCP, there is no distinction database servers in our system. By using a variation of Shamir's Secret Sharing scheme and treating all the databases equally, our system allows load balancing.

Ultimately, both DCP and our system only make stealing passwords more difficult on the server side. The attacker can still steal the password directly from the user. However, recent security breaches have shown that guarding credential databases against attacks is becoming an important aspect of protecting users' credentials.

References

[1] RSA Security: *Distributed Credential Protection (DCP)*
http://www.emc.com/collateral/software/white-papers/h11013-rsa-dcp-0812-wp.pdf

[2] Shamir, Adi: *How to Share a Secret*
http://dl.acm.org/citation.cfm?doid=359168.359176