

# Pvault: A Client Server System Providing Mobile Access to Personal Data

Ravi Chandra Jammalamadaka, Sharad Mehrotra, Nalini Venkatasubramanian  
Donald Bren School of Information and Computer Science University of California, Irvine Irvine, CA 92697  
{rjammala, sharad, nalini}@ics.uci.edu

## ABSTRACT

In this paper we describe the design for the *Pvault* software, which is a personal data manager that stores and retrieves data from a remote untrusted data server securely. The major advantage of *Pvault* is that it allows users to access their personal data from any trusted remote computer. We will describe the issues and solutions for maintaining data confidentiality and integrity when the data is stored at the remote sever, since the server itself is untrusted. *Pvault* also prevents Phishing and Pharming attacks and we will describe the solutions for the same.

## Categories and Subject Descriptors

H.2.8 [Information Systems]: Database Management—*Database Applications*; H.3.2 [Information Systems]: Information Storage and Retrieval—*Information Storage*; H.3.3 [Information Systems]: Information Storage and Retrieval—*Information Search and Retrieval*

## General Terms

Security

## Keywords

database, security, encryption, cryptography, secure sharing, mobile access, untrusted service provider model, secure storage

## 1. INTRODUCTION

Widespread acceptance of Internet as a medium for doing business has led to the emergence of many web services. Services such as email, news, online shopping portals, online banking ... etc have become quite popular with the internet savvy users. Such services are considered more the norm than a privilege for a select few, as most of the services are free for the user. All the services require users to authenticate themselves using usernames, passwords and/or other

personal information such as social security numbers. This scenario has led to users generating lot of personal data and that data is hidden in diverse systems across diverse software. Take for instance popular auto-fill feature available in many current browsers. This feature stores information entered in web forms, remembers it and fills out the web forms when they are visited in the future. Users have to fill out web forms only once and needless to say this is very popular convenient feature. When the users switch to a different computer or a different browser the auto-fill information is not available. Other examples of information that users want to be mobile include credit card numbers, pin numbers ... etc. This information is nothing but the identity information of the user and there is a genuine requirement for mobility/transportability of the identity information securely. Currently, users carry around their identity information in USB drives for achieving mobility. Such a solution is both insecure and inconvenient.

In this paper we propose a client-server architecture, similar in spirit to Database as a Service(DAS) model, as solution to the above problem. In this architecture, a client outsources its personal data to a service provider who provides the data management services such as storage and data access. Advantages of this solutions include a) lower cost, since the service provider can amortize the cost across clients b) mobile access to data, the client can connect to the server from any trusted computer c) facilitation of data sharing across organizational boundaries. There are two different types of security challenges that need to be addressed here a) there is a distinct possibility of hackers/internet thieves breaking into the service provider interface b) there is the concern of trust in the service provider itself. An approach to preserve the privacy of the client is through encryption. The client can encrypt all its data before outsourcing the data to the server/service provider. This further raises more challenges which include a) processing queries over encrypted data b) enabling data sharing/access control on encrypted data.

Recently, there has been an emergence of sophisticated online scams called *Phishing* and *Pharming* [3]. In a Phishing scam, the user typically receives a spoofed email from a financial institution urging it to update its personal information. The email's content generally contains a link which directs the user to a spoofed website that contains a web form. This website looks exactly like the original website the user relates to, although hosted on a different domain. Unsuspecting users can then enter their information and potentially face dire consequences. In a Pharming scam, an at-

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

StorageSS'05, November 11, 2005, Fairfax, Virginia, USA.  
Copyright 2005 ACM 1-59593-223-X/05/0011 ...\$5.00.

tacker exploits a vulnerability on a DNS server and changes the content of the directory which contains the domain to IP directory. Therefore, now a user can be directed to a spoofed website whenever a request to original website is forwarded to the DNS server. Similar to Phishing scam, an unsuspecting users can give away their personal information and become a victim of identity theft.

In this paper we describe the design of the Pvault system, which is a versatile secure system that can hold people's secrets. The Pvault system supports the following features:

- Supports mobile access to secrets from anywhere, anytime and from any machine.
- Generates strong passwords and helps registration of these passwords with websites.
- Remembers and fills out auto-fill information.
- Sharing of secrets in a secure, controlled fashion.
- Prevents Phishing and Pharming attacks.

The rest of the paper is organized as follows, in section 2 we describe our related work. In section 3 we describe the overall architecture of the Pvault system. In section 4 we describe the fundamental unit of data in the Pvault system, the Pvault entry. In section 5, we illustrate the main functionality provided by the Pvault client. In section 6, we describe the functionality provided by the service provider. In section 7 we list the weaknesses of the Pvault system. Section 8 deals with future work. In section 9 we conclude and in section 10 we acknowledge the contributors to the building of the Pvault system.

## 2. RELATED WORK

There are many commercial systems/research prototypes that offer some of the functionality of Pvault. Software for managing passwords [1, 2] are available on the internet. These softwares store encrypted passwords that belong to the user on the local machine. The *Autofill* feature, where the passwords are automatically filled for the websites the user visits, is available in most of these applications thereby, reducing the interaction between the user and the application. But all these applications do not provide the mobility of data since the data is stored encrypted on the local machine. The user is forced to carry the sensitive data manually with himself for achieving mobility. This further leads to problems with synchronization when data is updated on many other machines.

pwdhash[4] is a browser pluggin which generates a strong password for a website by hashing a master password with the domain name of the website. The user now has to remember only the master password and a unique password is generated on the fly everytime a user visits a website. This simple scheme gives users mobility with regard to their passwords as passwords can be generated on any trusted machine. The Drawbacks of this scheme are as follows a) Cannot store secrets other than passwords such as credit card numbers and pin numbers b) Updates to passwords(belonging to websites) are an issue, since a single update forces a change to the master password thereby requiring updates to all other passwords c) Since pwdhash generates passwords rather than store them, a new user is forced to register the new passwords generated with pwdhash.

AntiPhish[5] is a browser extension that tries to protect against Phishing attacks. AntiPhish works similar to the password managers describe above, where locally AntiPhish keeps track of sensitive information and warns the user whenever sensitive information is sent to a untrusted website. This tool has been implemented as an Mozilla extension.

Microsoft's single signon passport protocol[14] allows users to authenticate themselves to passport servers(microsoft run) to procure the required credentials for logging into participating websites. The drawbacks for this ambitious service are a) Requires changes in the authentication protocol of the participating website b) Requires complete trust in the passport service, since data is kept in plaintext in the passport servers, thereby vulnerable to insider/outsider attacks.

Pvault distinguishes itself from the above systems in the following manner a) It is based on the data outsourcing model b) Allows mobile access to secrets from any trusted remote machine c) Stores arbitrary secrets not just passwords.

## 3. ARCHITECTURE OF PVAULT

In this section we will briefly describe the overview of the Pvault service architecture. The architecture of the Pvault system is shown in fig 1. The two major entities in the Pvault architecture are the user/client and the service provider/server. The clients use the Pvault software to store and retrieve personal data(secrets). The fundamental data unit being outsourced by the client is a Pvault entry. A Pvault entry, similar to XML documents, consists of both structure and content. A Pvault entry besides storing personal data, optionally can store the required URLs (web pages) to which the personal data pertains to. The Pvault entry is described in more detail in section 4.

The users encrypts all the Pvault entries before outsourcing them to the service provider. We used Blowfish encryption algorithm[12] for encryption/decryption. The service provider now stores the Pvault entries and provides access to the entries whenever the client requests them using a conventional off the shelf relational database. The user has to authenticate himself with the server by supplying a username and password which the server can recognize. This is the username and password that user submitted to the server during the registration process. In the current implementation of pvault the password supplied to the server is the master password of the client encrypted with itself. Master password is the only secret the user has to remember for using Pvault. The master password is used as the key during encryption/decryption of clients secrets. The master password has to be strong, otherwise the user's data is vulnerable to dictionary/brute force attacks. The server then checks the set of <username,password> pairs available with it and verifies if the request to login from a client is legitimate. The server then transports all the sensitive data stored at the server to the client(This not necessarily is the case, but for simplicity, now we will assume that all the secrets are transferred). The Pvault client then decrypts the sensitive data using the master password.

Once the secrets are transported back to the user, the autofill feature is activated. Whenever, the user visits a website whose sensitive data is already stored as an pvault entry, the Pvault client will automatically fill out the required secrets(personal data) on the webpage. This feature is very important, since an advanced users of Pvault will use the

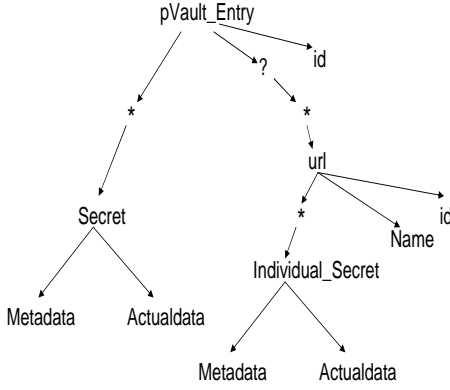


Figure 2: Schema of the Pvault entry

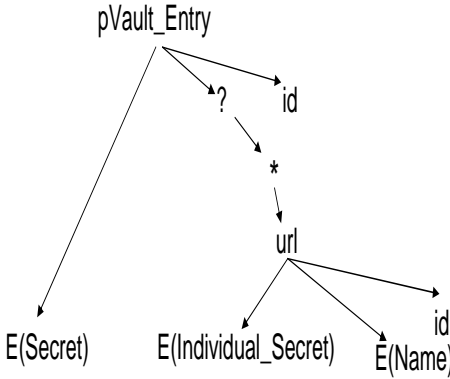


Figure 3: Schema of the encrypted Pvault entry

password generation feature to generate strong passwords for every website they visit. Since the user cannot be expected to remember the strong passwords, which is the one of the motivations for creating Pvault in the first place, there is the requirement for automatically filling the passwords on websites. Besides storing and retrieving secrets, Pvault also prevents Phishing and Pharming attacks. Details of how Pvault prevents the attacks will be explained in the coming sections.

#### 4. PVAULT ENTRY

In this section we will describe in detail about the fundamental data unit of the Pvault system, the Pvault entry.

The Pvault entry data model is based on the XML data model. All the Pvault entries are XML documents which follow a particular schema. XML documents are composed of nested elements, where each element is delimited by a start tag(eg. `<secret>`) and an end tag(eg. `</secret>`). The schema for an XML documents provides a formal definition on the structure of the XML documents. The schema of the Pvault entries is shown in fig 2. Operators( eg, \*,? ...

```

1.< Pvault_entry >
2.  < secret >
3.    < metadata > username < metadata >
4.    < actualdata > rav5678 </actualdata >
5.  </secret >
6.  < secret >
7.    < metadata > password < metadata >
8.    < actualdata > $%sdfg#&34FG </actualdata >
9.  </secret >
10. < url >
11.  < name > www.yahoo.com </name >
12. </url >
13. < url >
14.  < name > www.rediff.com </name >
15. </url >
16.</Pvault_entry >

```

Figure 4: Example Pvault entry

etc) are used to describe the cardinality of the child nodes for a particular node. “\*” operator implies “zero or more occurrences” and “?” operator implies zero or one occurrences. A Pvault entry could be potentially used to store secrets(personal data) that belong to more than one URL. For instance, the users are increasing maintaining profiles with websites, which usually includes information such as address, full name, email address ... etc. Users can maintain the same profile with more than one website. The Pvault entry schema in fig 2 states the following a) there could be zero or more secrets in a Pvault entry b) each Pvault entry can be associated with more than one URL c) and each URL can contain more than one secret. All the common secrets that belong to all the URLs are stored as the content of the secret element. Secrets that belong only to one url are stored as the content of the individual-secret element. The Optional operator “?” states that a Pvault entry may or may not contain URLs, since Pvault entries can store secrets that are not related to the web. The name element which is child of the URL element stores the url names. An example Pvault entry is shown in fig 4. We will now explain the details of the secret element.

A secret is denoted by using a `<metadata,actualdata>` pair. The *actualdata* element in the Pvault entry stores the actual value of the secret. The *metadata* element stores information which describes/provides information about the value stored in the *actualdata* element. We will now give an example about the data stored as the value of the *metadata* element.

Webpages contain a web form created using `<form action = URL>` tag, when user input is desired. Each form could contain a list of inputs elements which take in user input. The tag for an input element in HTML is `<input>`. Consider the following example of the `<input>` tag:

```
<input type = “password” value = “secretkey”>
```

The above HTML code creates a password input textbox on a webpage. The value attribute in the `<input>` tag “secretkey”, describes the password textbox. This information is stored as the value of the metadata element. The actual password entered by the user is stored as the value of the actualdata element. The metadata stored along with the actual secrets serves as association information which will guide Pvault to fill out the secrets at the appropriate place, when the webpage is visited again. Therefore, in future if

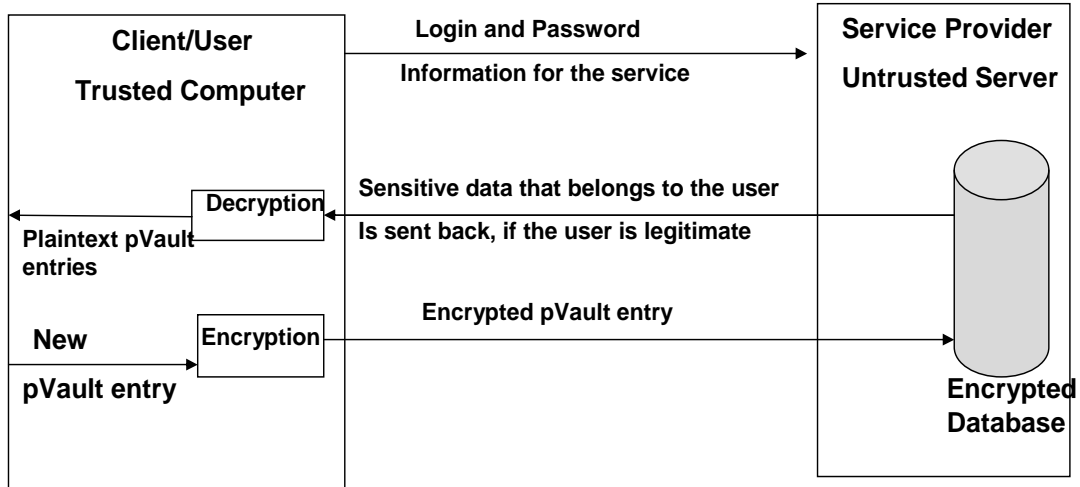


Figure 1: Pvault Architecture

the same url is visited again and Pvault encounters a password textbox, that has a value attribute to be “secretkey”, it will fill out the password textbox using the contents of the actualdata element stored in the Pvault entry.

We have till now explained the client side representation of the Pvault entry. All Pvault entries are created by the Pvault client requiring little manual input from the user. Before outsourcing the Pvault entries, the user needs to encrypt the Pvault entries to protect his privacy. One approach to encrypting the Pvault entries is to encrypt the values of the metadata, actualdata and name elements individually. In our approach, we encrypt the secret subtree and replace the subtree with an encrypted node  $E(\text{secret})$ . The content of the secret element (which is the whole subtree rooted at secret) is encrypted and placed as the content of the  $E(\text{secret})$  node. Similarly, the subtree rooted at `Individual_secret` is replaced by encrypted node  $E(\text{Individual\_secret})$ . This is being done to optimize the encryption/decryption times. If the metadata element and the actual element is encrypted individually, then when these elements are shipped back to the client, they have to be decrypted individually decrypted. All encryption algorithms have a startup time, therefore it makes sense to do bulk encryption/decryption. We have chosen not to replace the url subtree with an encrypted node. Depending on user preferences, only a secrets belonging to some URLs can be downloaded to the client. If the URL subtree is encrypted, then a selection based on name element cannot be made. Therefore, the name element is individually encrypted. The name element can be left as plaintext if the user wishes to do. Final structure of the encrypted Pvault entry is illustrated in fig 3.

## 5. PVAULT CLIENT

In this section we will describe the main functionality of the Pvault client.

### 5.1 Query Interface

Once the Pvault entries are downloaded from the server, the Pvault client allows the user to query the Pvault entries. When the user visits a webpage, Pvault checks if the information pertaining to the webpage is already available locally, if it is available locally, the data required is auto-

matically filled. If the data required for the webpage is not available locally, the user is prompted to check if it wishes to fetch the required data from the server. If the user wishes to do so, then the data is fetched from the server.

### 5.2 Password Generation

Pvault has a password generation tool which generates strong passwords with high entropy. It is widely known that passwords chosen by users have low entropy and therefore vulnerable to dictionary attacks. Pvault generates strong passwords of a given length (usually, given by a user or a system default is taken) by random choosing characters from a alphabet which includes numeric characters and special characters. Pvault creates strong passwords and helps the user to register these passwords during the registration process with a new website.

### 5.3 Autofill

Autofill is a convenient feature which allows users not to worry about filling out forms on the websites. As described before, once the Pvault entries are downloaded, the autofill feature is activated. When the user visits a new webpage, the Pvault client will try to find if there any Pvault entries that correspond to the URL. If multiple Pvault entries qualify, the Pvault client will prompt the user to disambiguate. This is usually done by the user clicking on the right pvault entry. For instance, such a problem can occur when a user has two accounts for a website, then Pvault will be unsure which account to fill out on the webpage. When the right Pvault entry is found, all the secrets that belong to the webpage are filled out. Recall that secrets stored in Pvault entries contain metadata and the actual data. Metadata guides the Pvault client software to find the appropriate input element in the HTML code to fill out the actual data in. Autofill feature does not guarantee that appropriate secrets are filled out in the required place, since the HTML code/design can be changed anytime by the webmaster of the website. Therefore, when Pvault does not correctly fill out a webpage, the user can correct Pvault mistakes. The Pvault entry pertaining to the webpage is then adjusted to make sure that in the future the mistakes do not happen again.

## 5.4 Phishing attacks

Phishing attacks are online scams where an adversary tries to steal personal information such as passwords, credit card numbers, social security numbers and other financial information from an unsuspecting user. These online scams are receiving much attention these days purely due to their frequency and their effectiveness. According to a recent study, 57 million US Internet users have recognized emails linked to Phishing scams, and about 1.7 million of them have given their personal data to these scams.

In a typical Phishing attack, the adversary sends a spam email to random users pretending to come from a financial institution such as a bank or from a website which offers some popular service. The email urges the user to update its personal information under some false pretext which is usually a security upgrade. This email will contain a link which directs the user to a website which has a form for the user to fill. This website is a spoofed website which looks and feels exactly like the real website the user is accustomed to. An unsuspecting user could then enter its personal information. Needless to say, this attack is very effective with potentially catastrophic consequences for the poor user. Recently, Phishing attacks are becoming more sophisticated, where emails are sent to individual users not necessarily spam to many users. This email contains individual information that does belong to the user such as an account number, date of birth ... etc. Since the email has some personal information, users are more likely to believe that this email is legitimate and send personal information to the spoofed website.

For a careful user who checks the domain names of all the urls that he/she visits, Phishing attacks are not a problem, since the attack can be manually caught. But to expect every user to pay careful attention to every website he/she visits is unreasonable especially, when the spoofed websites are so close to the websites that users relate. With the Phishing attacks getting more sophisticated, it is no wonder that these attacks are so successful. We need solutions/applications that prevent the user from submitting information to malicious websites. Pvault prevents Phishing attacks by analyzing the information being submitted to every website. Pvault requires that user store sensitive data pertaining to a websites such as passwords to be stored with it. Whenever a user submits information to a webpage form, Pvault captures this event and compares this information with its local database. If the information being submitted to the website belongs to another website, Pvault will raise an alert. It is now for the user to decide to continue to send the information or not by clicking the Yes/No buttons on the alert message box.

## 5.5 Pharming

Pharming attacks are online scams where an attacker/adversary tries to exploit vulnerability in the DNS server. The DNS server is responsible for mapping domain names to IP addresses. Every DNS server maintains a directory which essentially is a table that maps domain names to IP addresses. The attacker tries to poison this table or change the contents of it, thereby procuring the ability to redirect users to a spoofed website rather than the original one. This is far more sophisticated scam than Phishing attack that we described before, since just checking the url will not help here. To prevent this attack Pvault requests domain name

1. Alice Generates a random key  $k$ .
2. Encrypts Pvault entry  $PE$  with key  $k$  which generates  $E_k(PE)$
3. Requests the Bob's public key  $B_{pu}$  from the server.
4. Encrypts key  $k$  with Bobs public key  $E_{B_{pu}}(k)$
5. Stores  $E_k(PE)$  and  $E_{B_{pu}}(k)$  with Bob's account in the server

Figure 5: pVault entry sharing protocol

to IP address conversion to not one but three domain name servers spread around the world. If all the three IP addresses returned do not match then Pvault detects a possible Pharming attack. This solution will not work in the highly unlikely event of all three DNS servers being poisoned and all of them returning the exact IP address for a domain name. Even though the solution given here is not complete, nevertheless it is practical. We have chosen to query three DNS servers, although more than three can be queried, but it comes at a cost since it will slow down the system.

## 5.6 Secure Sharing of Pvault entries

Pvault allows secure sharing of Pvault entries between users. The sharing protocol is based on the Public key Infrastructure(PKI). When a new user registers with the Pvault, a <public key,private key> key pair is generated and the public key is stored/registered with the server. The private key is stored with the Pvault client. Fig 5 describes the sharing protocol. Alice a user of Pvault is try to share her Pvault entry  $PE$  with Bob another Pvault user.

Alice generates a random key  $k$ , using which she encrypts the Pvault entry that she wants to share. Here we use symmetric encryption, where the encryption and decryption are done with the same key. Alice then requests the server for Bob's public key and encrypts the key  $k$  with Bob's public key. She then instructs the server to store both the encrypted Pvault entry and the key  $k$  encrypted with Bob's public key  $E_{B_{pu}}(k)$ , in Bob's account. Bob can now decrypt  $(E_{B_{pu}}(k))$  using his private key to find the key  $k$  used to encrypt the Pvault entry  $PE$ . Now since Bob has the key  $k$ , he can then decrypt  $E_k(PE)$  to access the Pvault entry that has been sent by alice. A copy of the Pvault entry  $PE$  is now stored in Bob's account, with the server. All the above protocol is executed by the Pvault clients at both Bob's and alice side. Alice just has to instruct the Pvault client to share the appropriate  $PE$  with Bob and Bob has to accept the request for sharing from alice.

## 6. PVAULT SERVER

In this section we will describe the major functionality of the server in the Pvault architecture. We will be limiting our discussion to the following a) How are the encrypted Pvault entries represented/stored at the server b) How to prevent server from tampering with the outsourced data.

The Pvault clients outsources encrypted Pvault entries which are XML documents. These XML documents are stored at server using a standard off the shelf relational database. The encrypted documents are shredded into relational tuples before being stored. We have used the *inlining technique* proposed in [9] to store the XML documents in relational tables. In this technique, a table is created for the root element and for every element that is under

a “\*” operator in the XML schema. The other elements are inlined/included as columns with the nearest ancestor that has a table created for it. The relational schema at the server side that stores XML documents that follow the Pvault entry schema in fig 3 is as follows:

*PVAULT\_ENTRY(id, etuple)*  
*URL(id, etuple, parentid, ename)*

The *PVAULT\_ENTRY* table stores the *id* of the Pvault entry in plaintext and *etuple* attribute stores the content of the *E(secret)* node. Recall that *E(secret)* stores all the common secrets that belong to all the urls in encrypted form. The *URL* table stores a tuple for every url that belongs to a Pvault entry. Every url is given a unique id which is stored in the *id* attribute, *etuple* attribute stores the content of the *E(Individual\_secret)* node. The *ename* attribute stores the encrypted URL name. The *parentid* attribute is the id of the Pvault entry to which the url belongs to. This is necessary to select the URL table tuples that belong to a Pvault entry.

Furthermore, the encrypted information stored at the server can also include some indexing information to enable to search over encrypted data. Using the indexing information a superset of results can be retrieved that can be sent back to the client. The client can then filter the unwanted results after decryption. Such techniques have been previously explored in [10, 13, 12, 11] to execute SQL queries over encrypted data. Exploiting such techniques in our context adds the following challenges a) the techniques proposed in the literature require metadata to be stored at the client, which made the client side application fairly thick/ substantially large b) the techniques were proposed for outsourced database model, where the databases were very large, unlike our case where we are concerned about personal data which is considerably less in size c) the metadata stored at the client will interfere with the sharing protocols. Enabling search over encrypted data has high priority for our future work and this feature has not been implemented in our current version of Pvault system.

## 6.1 Maintaining Data Integrity

In this section we will describe how the client/user can make sure that the data exported/outsourced to the server is not tampered with. The server can a) delete data b) make changes to the data c) insert new data. We will describe cryptographic techniques using which can client can detect if any changes have been made at the server side to its data. We are essentially trying to guarantee both the *soundness* and the *completeness* of the data sent from the server.

### 6.1.1 Guaranteeing Soundness

Recall that the client stores Pvault entries at the server. Together with the Pvault entries, message authentication codes (MACs) are computed and stored along with every Pvault entry. The MAC generation algorithm which is a cryptographic hash function, accepts the input message and a secret key and produces the message digest or the MAC. The input message here is the concatenated string of all the leaf node values of the Pvault entry and the secret key is the master password. When the server sends the Pvault entries back to the user, the servers also sends the MACs back to the user. The user recomputes the MAC for every Pvault entry and compares it with the MAC received from the server. If the MACs match, then the the Pvault entry

is *sound*. MACs are efficient to compute and therefore are not a huge performance impediment.

### 6.1.2 Guaranteeing Completeness

In the previous section we have shown how we guarantee soundness of the Pvault entries sent back to the user. But guaranteeing soundness is only one part of the problem, how do we guarantee that the Pvault entries sent back to the client are *complete*. The server can send only a subset of the Pvault entries or insert some fake Pvault entries (which could be a copy of a existing Pvault entries i.e a duplicate). A count of all the tuples can be maintained at the server encrypted with the master password together with its MAC, which prevents the server from manipulating the count itself. The count can be sent back to the user, who can then verify how many sound Pvault entries have been sent back. But this does not solve the problem of receiving duplicate Pvault entries. To ensure that user does not receive duplicate entries, every Pvault entry is assigned a unique id(Refer to fig 2). Recall that id is used during MAC generation of the Pvault entry. Now the duplicates can be easily caught at the user side.

## 7. WEAKNESSES OF THE PVAULT SERVICE

In this section we will list some of the drawbacks of the Pvault software/service.

**One point of failure:** All the Pvault entries are guarded by one master password. If the master password is compromised, all the Pvault entries are easily known to the adversary. It is important that users choose a strong password as the master password. It is recognized that passwords chosen by users have low entropy and are easily susceptible to dictionary attacks. Recently authors of [6] have proposed techniques for strengthening user passwords, by repeatedly applying cryptographic hash functions on passwords. Similar techniques could be applied on the master password to further strengthen the security.

**Pvault client needs installation:** The current implementation of the Pvault client requires users to install the software on their local machine. Users have remote access to their data as long as the computer has Pvault software. When users are trying to access their passwords from a cybercafe, it may not be possible to install software. Therefore, to allow remote access to their data from any remote computer, we are presently working on a web based implementation of Pvault.

## 8. FUTURE WORK

Pvault is currently only supported on Windows platform and the autofill feature only works with Internet explorer. The reason for choosing the Windows platform as the operating system and Internet explorer as the browser, is their popularity. In the future, we will make Pvault supported on other platforms such as Linux, Solaris and on browsers such as Mozilla which is gaining popularity quickly. We have already stated that we are currently working on web based implementation of the Pvault software. Even though tasks such as encryption/decryption can be implemented on browser based scripting languages such as Javascript, certain features such as autofill cannot be implemented using web technology.

Currently, Pvault service stores only secrets/personal data of the users. In future Pvault will evolve into a tool that allows users to share data. Users can outsource complex data objects to the service provider and can enforce access control to these objects through cryptography. The advantages of such a solution include reduction of operation costs, mobile access to data and facilitation of data sharing.

## 9. CONCLUSIONS

In this paper, we presented the Pvault software application. Pvault allowed users to outsource personal data to an untrusted server. Data confidentiality and Integrity were preserved using cryptographic techniques. Pvault system allowed users seamless mobile access to their personal data. Pvault autofill feature fill outs passwords/other information on websites, thereby relinquishing users of the responsibility. Pvault also prevents online scams such as Pharming and Phishing. Our system is available on the internet. The released version of Pvault does not have the following features described in this paper a)Secure sharing of Pvault entries b)Pharming c) Maintaining data integrity. These features are concurrently under development and a stable version of Pvault with these features will be released soon. We encourage users to try it and give us some feedback. The Service has been serving 20 beta testers for more than 6 months now. Pvault is currently available for download here [7].

## 10. ACKNOWLEDGEMENTS

We would like to thank the privacy team at University of California, Irvine, comprising of Bijit Hore, Jehan Wekramasinghe and Mahesh Datt (the list does not include the authors who also belong to the team), for helpful comments on Pvault. The authors are indebted to Yonghua Wu for his help in implementing the Pvault system. Chris Davison(Pvault administrator) has worked tirelessly to make sure that Pvault server is running 24/7. Special thanks to Einar Noronha Mykletun for providing the source code for Blowfish encryption algorithm. We thank the anonymous reviewers for providing helpful suggestions for improving the quality of the paper.

## 11. REFERENCES

- [1] Password Vault.  
<http://www.rit.edu/smo5024/projects/pvault/>
- [2] Password Safe.  
<http://www.schneier.com/passsafe.html>
- [3] Anti-Phishing Working Group.  
<http://www.antiphishing.org>
- [4] Blake Ross. Collin Jackson, Nicholas Miyake, Dan Boneh and John C. Mitchell Stronger Password Authentication Using Browser Extensions. *To appear in Proceedings of the 14th Usenix Security Symposium, 2005.*
- [5] Engin Kirda and Christopher Kruegel. Protecting Users Against Phishing Attacks with AntiPhish. *In Proceedings of the 29th Annual International Computer Software and Applications Conference (COMPSAC), IEEE Computer Society Press. United Kingdom, July 2005.*
- [6] J. Alex Halderman, Brent Waters, Edward W. Felten. A Convenient Method for Securely Managing Passwords. *In Proceedings of the 14th International World Wide Web Conference (WWW 2005).*
- [7] pVault Homepage.  
<http://www.itr-rescue.org/pVault/>
- [8] B. Schneier. Description of a New Variable-Length Key, 64-Bit Block Cipher (Blowfish). *Fast Software Encryption, Cambridge Security Workshop Proceedings (December 1993), Springer-Verlag, 1994, pp. 191-204*
- [9] Jayavel Shanmugasundaram, Kristin Tufte, Chun Zhang, Gang He, David J. DeWitt, Jeffrey F. Naughton: Relational Databases for Querying XML Documents: Limitations and Opportunities. *International Conference on Very Large Databases (VLDB 1999): 302-314*
- [10] Hakan Hacigumus, Bala Iyer, Chen Li, and Sharad Mehrotra. Executing SQL over Encrypted Data in the Database-Service-Provider Model. *2002 ACM SIGMOD Conference on Management of Data, Jun, 2002.*
- [11] Ravi Jammalamadaka, Sharad Mehrotra. Querying Encrypted XML Documents. *University of California, Irvine. Technical report TR-DB-04-03.*
- [12] Bijit Hore, Sharad Mehrotra, Gene Tsudik. A Privacy-Preserving Index for Range Queries. *International Conference on Very Large Databases (VLDB 2004), Toronto, Canada 2004.*
- [13] H. Hacigumus, B. Iyer, and S. Mehrotra. Ensuring Integrity of Encrypted Databases in Database as a Service Model. *IFIP Conference on Data and Applications Security, Estes Park Colorado, 2003.*
- [14] Microsoft Passport Network.  
<http://www.passport.net>