Privacy Preserving Identity Attribute Verification in Windows CardSpace

Kevin Steuer Jr
CS Department
Purdue University
West Lafayette, Indiana
ksteuer@cs.purdue.edu

Ruchith Fernando
CS Department
Purdue University
West Lafayette, Indiana
rfernand@cs.purdue.edu

Elisa Bertino
CS Department
Purdue University
West Lafayette, Indiana
bertino@cs.purdue.edu

ABSTRACT

There are various identity management systems in place today. The way these share information about entities that interact with them gives rise to various privacy issues. This paper addresses two main such issues in Windows CardSpace, regarding the trust users' have to place on the system with respect to their personal information and to protect them against being exploited by means such as profiling. The protocol extensions proposed were implemented with a prototype as a proof of concept.

Categories and Subject Descriptors

D.4.6 [**Operating Systems**]: Security and Protection; K.6.5 [**Management of Computing and Information Systems**]: Security and Protection

General Terms

Security

Keywords

Digital Identity Management, Identity Metasystem, Windows CardSpace, VeryIDX, Linkability

1. INTRODUCTION

With the wide spread use of various on-line services, it is immensely important to protect our *digital identities* against a plethora of attacks, where users' [1] personal information can be easily compromised. A digital identity of a user can be defined as a set of statements about the user. These statements, called *claims*, are based on properties (identity attributes) (attributes, for short) of the user. An example of such a property is the Social Security Number issued by the United States government.

[1]By the user we mean the individual to whom the data refers to and that could be adversely affected by data theft or unauthorized disclosure.

The current implementations of Windows CardSpace however, have two major issues, that we address by integrating CardSpace with the VeryIDX [4] protocols. The VeryIDX system addresses these issues with a set of protocols supporting privacy-preserving verification of digital identities. These protocols allow users to prove to service providers the ownerships of their identity claims without revealing the contents of the identity claims.

The first issue is that the identity manager has to be completely trusted by a user to hold her identity attributes. Our extension to the Windows CardSpace protocols allows one to incorporate a *semi-trusted identity manager*, which does not require to store the attribute values locally at the identity manager, and allows the user to prove the knowledge of those attribute values to the relying party.

The second issue is that the CardSpace identity manager returns a signed set of attribute values to a relaying party about a certain user. This allows a relying party to profile the user's activities with respect to the services it provides. Furthermore, one or more relaying parties may collude and share information about the same user. We address this problem, known as the problem of linkability of digital identities, by another extension to the CardSpace protocols where the relying parties are be able to verify the user's knowledge of a certain identity attribute value without being able to establish relationships to previous uses of the service.

2. THE PROBLEMS AND SOLUTIONS OVERVIEW

In this section we discuss the amount of trust users have to place on the identity manager and introduce the problem of linkability at relying parties. We also outline our solutions to these issues.

2.1 Trusting the Identity Manager

Most identity managers store data about users for control and are fully trusted by the users to securely store this data. This approach has some obvious vulnerabilities and high costs. For example, even if data is encrypted when on the storage media, malicious insiders can still steal and misuse this information. Some current database management systems, like SQL Server, support fully transparent database encryption (TDE), by which the entire database, including indexes, logs, and catalogs, are encrypted; however, the TDE manuals and white papers clearly say that data is not protected by encryption when in main memory. If the identity manger infrastructure is compromised, for example by a rootkit or a Trojan inside the operating system, the memory can leak sensitive data.

It is important to notice that in many cases identity claims are simply needed to verify some information about a user. For example a relying party might want to know whether a user is a resident of a country. Therefore the attribute values that are used to derive the above information do not need to be available in clear to the relying party. It is sufficient that the derived information be available, or that the relying party be able to verify that the attributes verify certain conditions (for example, age > than 18). Even though Windows CardSpace already supports claim derivation, the user is still forced to let the identity manager store all identity attribute values.

2.2 Linkability

In Windows Cardspace, the identity selector interface is designed to have a user naturally use the same information card with the same relying party each time. For example the card that was used to access a service once will be highlighted for the user to select if the user accesses the same service again. Therefore the relying party will receive consistent claim values about the user from the identity manager. These can be used to track the user's behavior and profile her within the relying party. Even though this can be avoided by the user using a different information card each time a service is accessed, it not practical to do so. Furthermore, relying parties can collude to match claim values and profile a user. We present a technique for generating the cryptographic commitments to address this linkability issue.

3. PRELIMINARY NOTIONS

In this section we introduce the necessary background information needed for the presentation of our protocols. We first review the Pedersen commitment scheme and then the Schnorr's protocol for ZKPK as described by Paci et al. [5].

3.1 Pedersen commitment

The Pedersen Commitment scheme, first introduced in [6], is an unconditionally hiding and computationally binding commitment scheme that is based on the intractability of the discrete logarithm problem.[2]

Pedersen Commitment
Setup
A trusted third party T chooses a finite cyclic group G of large prime order p so that the *computational Diffie-Hellman problem*[3] is hard in G. Write the group operation in G as multiplication. T chooses an element $g \in G$ as a generator, and another element $h \in G$ such that it is hard to find the discrete logarithm of h with respect to g, i.e., an integer α such that $h = g^\alpha$. T may or may not know the number α. T publishes G, p, g and h as the system's parameters.

[2]Let G be a (multiplicatively written) cyclic group of order q and let g be a generator of G. The map $\varphi : \mathbb{Z} \rightarrow G, \varphi(n) = g^n$ is a group homomorphism with kernel $\mathbb{Z}_{||}$. The problem of computing the inverse map of φ is called the *discrete logarithm problem (DLP) to the base of g*.

[3]For a cyclic group G (written multiplicatively) of order q, with a generator $g \in G$, the *Computational Diffie-Hellman Problem* is the following problem: Given g^a and g^b for randomly-chosen secret $a, b \in \{0, \dots, q-1\}$, compute g^{ab}.

Commit
The domain of committed values is the finite field $\mathbb{F}_p$ of p elements, which can be represented as the set of integers $\mathbb{F}_p = \{0, 1, \dots, p-1\}$. For a party U to commit a value $x \in \mathbb{F}_p$, it randomly chooses $r \in \mathbb{F}_p$, and computes the commitment $c = g^x h^r \in G$.
Open
U shows the values x and r to open a commitment c. The verifier checks whether $c = g^x h^r$.

3.2 Zero-knowledge proof of knowledge (ZKPK) protocol

In the Pedersen commitment scheme described above, a party U referred to as the prover, can convince the verifier, V, that U can open a commitment $c = g^x h^r$, without showing the values x and r in clear. Indeed, by following the zero-knowledge proof of knowledge (ZKPK) protocol below, V will learn nothing about the actual values of x and r. This ZKPK protocol, which works for Pedersen commitments, is an adapted version of the zero-knowledge proof protocol proposed by Schnorr [7].

Zero-knowledge proof of knowledge (Schnorr protocol)
As in the case of Pedersen commitment scheme, a trusted party T generates public parameters G, p, g, h. A prover U who holds private knowledge of values x and r can convince a verifier V that U can open the Pedersen commitment $c = g^x h^r$ as follows.

1. U randomly chooses $y, s \in \mathbb{F}_p^*$, and sends V the element $d = g^y h^s \in G$.
2. V picks a random value $e \in \mathbb{F}_p^*$, and sends e as a challenge to U.
3. U sends $u = y + ex, v = s + er$, both in $\mathbb{F}_p$, to V.
4. V accepts the proof if and only if $g^u h^v = d \cdot c^e$ in G.

4. PROPOSED EXTENSIONS

In this section we introduce a new type of information card, referred to as *VeryIDX managed card* (VMC). Such a card is used to convey claims of a special type, called *strong claims*. The VeryIDX managed cards are used in cases where the relying party expects the user to prove the knowledge of an attribute, and does not require to see the attribute in clear. The information that the user proves knowledge of is asserted by the identity manager. When using the VeryIDX managed card, the parties will carry out our Identity Attribute Verification Protocol. In what follows, we first present our extensions to the information card format and then our protocols. Finally, we describe how to address the linkability problem while proving knowledge using the above approach.

4.1 VeryIDX Managed Card

The VeryIDX managed card is structured as a conventional information card with the addition of a new element, referred to as $SupportedStrongClaimValues$.
The $SupportedStrongClaimValues$ element is a collection of $StrongClaimValue$ elements each representing a strong claim. Each$StrongClaimValue$, in turn, contains three elements: a commitment value c, a random value r, and an expiration date exp. c is a Pedersen commitment and is

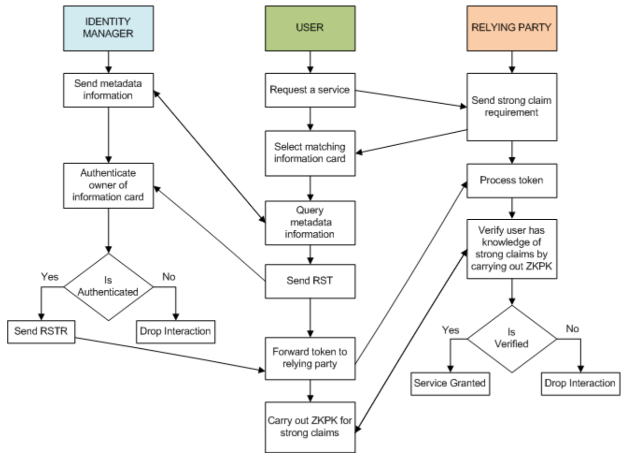
IDENTITY MANAGER
USER
RELYING PARTY
Send metadata information
Request a service
Send strong claim requirement
Select matching information card
Authenticate owner of information card
Process token
Query metadata information
Verify user has knowledge of strong claims by carrying out ZKPK
Yes
Is Authenticated
No
Send RST
Send RSTR
Drop Interaction
Yes
Is Verified
No
Forward token to relying party
Service Granted
Drop Interaction
Carry out ZKPK for strong claims

Figure 1: Identity Attribute Verifcation Protocol

generated using the attribute value x and a random value r by running the Commit protocol introduced in section 3.1. We assume that the identity manager forgets the two values x and r after the initial commitment generation leaving the random value r only known to the identity selector and attribute value x only known to the user. The commitment c is used at runtime to provide proof of knowledge of the claim value without the other parties learning any information about x or r. The expiration date, exp, is an optional parameter to allow the identity manager to invalidate the commitment after a given time period.

Suppose that identity manager *IDM* issues a VMC to user U who uses identity selector *IS*, containing a strong claim generated using an attribute value x. Let (G,p,g,h,$K_{IDM_{Pub}}$) be the public parameters of *IDM*. $K_{IDM_{Pub}}$ and is the public key of *IDM* and $K_{IDM_{Priv}}$ is the corresponding private key. *IDM* generates a VMC containing a $SupportedClaimType$ element that is a claim of type strong claim.
There is a corresponding $StrongClaimValue$ element within the $SupportedStrongClaimValues$ element. In Windows CardSpace, the *SupportedClaimType* elements are used to indicate the claim offerings supported by the information card. We refer to these objects as elements because they exist within the managed card's schema represented in XML [3]. The VMC is signed by $K_{IDM_{Priv}}$, that is, the private key of *IDM*. *IS* obtains the VMC, and verifies its authenticity and integrity using $K_{IDM_{Pub}}$ and stores it in the local card store for *U*. In Windows CardSpace the local card store is assumed to be secure.

Example 4.1. *Suppose an organization named* VerifySSN *provides a service to verify your* SSN *and picture ID in person and issue a VeryIDX managed card containing the associated commitment. Users with VeryIDX managed cards from* VerifySSN *can prove their United States resident status to relying parties while preserving their privacy and not revealing their actual* SSN*.*

Table 1 shows the logical organization of the StrongClaimValue records, securely stored in the identity selector's local card store, for strong claims related to the social security number and the AAA membership.

StrongClaimValue			
tag	commitment	random value	expiration
SSN	7438726487979	329798493827	1/1/2011
AAA ID	9863214562558	889587963452	1/1/2011

Table 1: Sample StrongClaimValues

4.2 Verification of Strong Claims

A user requesting a service from a relying party can prove knowledge of the strong claim attribute values contained in a VMC. Verification of strong claims is divided into three main phases. Figure 1 summarizes the steps in these phases.

1. Requesting a service that requires a strong claim.
2. Obtaining a token from the identity manager.
3. Providing proof of knowledge of strong claim values to the relying party.

Strong claim verification request

This phase starts when a user makes a request to a relying party providing some service. To use the service the relying party has a requirement to verify a set of strong claims. The relying party's claim requirements are conveyed the user's identity selector. The user's identity selector queries the local card store for possible matches and presents the user an interface to select the appropriate VMC containing the set of matching claims.

Obtaining a token from the identity manager

Once a user selects the appropriate VeryIDX managed card, the identity selector may prompt the user for the authentication credentials of the card. The identity selector sends the authentication information and the relying party's public key ($K_{RP_{Pub}}$) to the identity manager. For each strong claim the identity manager verifies its corresponding expiration date. If the value has expired the transaction is dropped and the user needs to obtain a new VMC or use another card. If the authentication and expiration verification succeeds, the identity manager constructs a response which contains a token for the relying party. This token is encrypted using the $k_{RP_{Pub}}$ and signed by $K_{IDM_{Priv}}$ so that the token cannot be modified in transit. The token contains all the information requested by the relying party. For each strong claim, the identity manager includes in the token the Pedersen commitment value c. The response is sent to the user's identity selector which then removes the token and forwards it to the relying party.

ZKPK with relying party

When the relying party receives the token from the user's identity selector, the token is decrypted using the relying party's private key and its authenticity and integrity is verified using $K_{IDM_{Pub}}$. The relying party checks that all required regular claim values are available. Then for each strong claim commitment value, the ZKPK protocol is carried out with the identity selector. For ZKPK the identity selector uses attibute values which are only known to the user and random values associated with the commitment which are located in the VeryIDX managed card. If the proof succeeds, the user is granted service.

The previous protocol does not address from the linkability problem. When the same attribute encoded in a strong claim is used multiple times, relying parties are able to profile the user because the commitment values included in the token do not change.

We address the problem by introducing a new ZKPK proof step along with the authentication carried out with the identity manager to prove the user's knowledge of the strong claim attribute values used to generate the VeryIDX managed card. In the new protocol, whenever a user needs to submit a strong claim for an attribute a to a relying party, the user carries on ZKPK with the identity manager. Upon a successful proof, the identity manager generates a new commitment which is sent to the identity selector which will then forward it to the relaying party and engage in a ZKPK proof with the relying party with this new commitment. It is important to notice that these changes are embedded in the identity selector and are fully transparent to the relying party and to the user. The user is prompted to enter the attribute values only once, that is, when carrying on the ZKPK with the identity manager. Therefore, the user experience is the same as in the previous protocol.

Let the initial commitment created during VeryIDX managed card issuance be $c_1 = g^x h^{r_1}$, where r_1 is the random value used in the Pedersen Commit protocol. After the user has proved knowledge of the attribute value x (which is used to generate c_1) during the authentication phase, the identity manager generates a new $c_i = g^{c_1} h^{r_i}$, where r_i is a new random number. c_i is sent to the relying party for ZKPK of c_1 and r_i to be carried out with the identity selector.

5. IMPLEMENTATION

We have implemented a prototype of our protocols as a proof of concept.

For the identity selector component, we considered using the Windows CardSpace selector. However the Windows CardSpace selector is not designed to be extensible. Therefore we settled on the Higgins identity selector, that supports the Windows CardSpace protocols. Higgins is an open source identity framework and we specifically used the Higgins 1.1 Gtk selector [1], which is implemented in C++. We implemented the ZKPK protocols and the Pedersen commitment libraries required at the identity selector in C++. To handle large integer values we used the NTL library [8].

The identity manager component we used is the WSO2 Identity Server which supports Windows CardSpace and the OpenID protocols. We modified the information card issuance of this identity manager to issue VeryIDX managed cards and its Security Token Service was modified to include the commitment values in the issued tokens. We used the relying party library available with the WSO2 Identity Server to setup a Java web application which required a strong claim. In the identity manager and the relying party components, we used the ZKPK library implemented in the VeryIDX project.

6. CONCLUSIONS AND FUTURE WORK

We have proposed an extension to Windows CardSpace for attribute verification that does not reveal any additional information to the relying party. This extension allows for the identity manager to be semi-trusted, different than most current systems today. This approach not only can reduce the liability of the identity manager but it also lets the user control how her information is used. We also provide the option in our extension to disable profile tracking from a relying party by randomizing commitments used during verification.

We plan to continue our work with CardSpace on the following issues: sharing information cards, the amount of trust in the identity selector, relaxing the trust requirement of the identity manager during the initial phase, and supporting derived values.

In our approach, we assume that, upon issuance of a VeryIDX managed card, the identity manager forgets the sensitive information used to generate the strong claim. As part of future work we plan to investigate approaches to relax this assumption by allowing the users to generate the commitments while ensuring the correct value is used.

Our protocols do not allow derived values on strong claim values. Derived values are used when verifying a conditional statement (e.g. if the user is older than 18). The Pedersen commitment can be used with the OCBE protocols [2] to address this issue. As part of future work we plan on investigating how our protocols can be extended with the OCBE protocols and other protocols for the verification of predicates on encrypted values.

7. ACKNOWLEDGEMENTS

This work was supported in part by AFOSR grant A9550-08-1-0260.

8. REFERENCES

[1] Higgins. Gtk selector 1.1-win. Available at `http://wiki.eclipse.org/GTK_Selector_1.1-Win`.

[2] J. Li and N. Li. Oacerts: Oblivious attribute certificates. *IEEE Trans. Dependable Sec. Comput.*, 3(4):340–352, 2006.

[3] A. Nanda and M. B. Jones. Identity selector interoperability profile v1.5. Technical report, Microsoft, 2008.

[4] F. Paci, E. Bertino, S. Kerr, A. Lint, A. C. Squicciarini, and J. Woo. Veryidx - a digital identity management system for pervasive computing environments. In U. Brinkschulte, T. Givargis, and S. Russo, editors, *SEUS*, volume 5287 of *Lecture Notes in Computer Science*, pages 268–279. Springer, 2008.

[5] F. Paci, N. Shang, S. Kerr, K. S. Jr., J. Woo, and E. Bertino. Privacy-preserving management of transactions' receipts for mobile environments. In K. E. Seamons, N. McBurnett, and T. Polk, editors, *IDtrust*, ACM International Conference Proceeding Series, pages 73–84. ACM, 2009.

[6] T. P. Pedersen. Non-interactive and information-theoretic secure verifiable secret sharing. In J. Feigenbaum, editor, *CRYPTO*, volume 576 of *Lecture Notes in Computer Science*, pages 129–140. Springer, 1991.

[7] C.-P. Schnorr. Efficient identification and signatures for smart cards. In G. Brassard, editor, *CRYPTO*, volume 435 of *Lecture Notes in Computer Science*, pages 239–252. Springer, 1989.

[8] V. Shoup. Ntl: A library for doing number theory. Available at `http://www.shoup.net/ntl/`.